Facultad de Ciencias

Nuevas herramientas de apoyo para la evaluación automática de ejercicios de programación en Moodle a través de la extensión EvalCode

(New support tools for automatic evaluation of programming exercises in Moodle through the EvalCode extension)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Ernesto Martínez Gutiérrez

Director: Héctor Pérez Tijero

Co-Director: Mario Aldea Rivas

Septiembre - 2020

AGRADECIMIENTOS

A mis padres, que nunca me presionaron ni condicionaron y siempre confiaron en mi buen criterio y decisiones en todos estos años.

A mis amigos de la facultad, que amenizaron mis días en la universidad y fueron una tremenda ayuda y apoyo para que acabase consiguiendo mis objetivos.

A la Universidad de Cantabria, y en especial al Dpto. de Ingeniería Informática, por permitirme ser parte de este proyecto que ha sido un gran complemento para mi formación universitaria y en el que he recibido un gran apoyo en todo momento.

RESUMEN

En un mundo cada vez más informatizado, las plataformas de gestión de aprendizaje online como Moodle cobran cada día más importancia y se están convirtiendo en piedra angular de la enseñanza universitaria, otorgando grandes posibilidades a nivel de evaluación con tareas, encuestas, test…. Especialmente en el caso de la informática y la programación, el aprendizaje continuo basado en la realización de tareas resulta clave, pero estas deben ser rigurosamente corregidas y revisadas por el docente, lo que puede resultar altamente costoso en términos de tiempo y esfuerzo. Por ello estas plataformas pueden ser mejoradas para la automatización de procesos de evaluación. En esta temática se enmarca el proyecto EvalCode, un proyecto que se inició en la Universidad de Cantabria hace tres años. Este software ayuda a los profesores integrando herramientas de corrección automática en Moodle como extensión de la plataforma. En este proyecto retomaremos el trabajo previo de anteriores versiones de EvalCode, realizando el mantenimiento de las herramientas de evaluación ya implementadas, e integrando nuevas funcionalidades, como una herramienta de corrección de estilo para lenguaje C, una herramienta tipo *sandbox* que nos garantice una ejecución segura de código externo, y una herramienta de detección de plagio. El objetivo final de este proyecto es que el software EvalCode, con todas sus funcionalidades, sea usado por alumnos de Ingeniería Informática de la Universidad de Cantabria.

Palabras Clave: Evaluación automática, Moodle, Programación, PHP

ABSTRACT

In a world more computerized by the day, the online learning management platforms like Moodle are becoming increasingly important as a cornerstone of university education, giving lots of evaluation possibilities with assignments, surveys, tests…. Especially in the case of informatics and programming, learning with continuous assignments is a key point, but the assignments should be strictly revised by the teacher, which may be highly costly in terms of time and effort. For this reason, these platforms can be improved with the automatization of evaluation processes. In this field is situated the EvalCode project, initiated three years ago in the University of Cantabria. This software helps the teachers by including automatic evaluation tools in Moodle as an extension of this platform. In this project we will resume the work of the previous versions of EvalCode, performing the maintenance of the tools that are already implemented and including new functionalities, such as a style correction tool for C language, a sandbox tool that guarantee a secure execution of external code, and a plagiarism detection tool. The purpose of this is EvalCode project is to be used, with all its functionalities, by the students of the Grade on Informatics Engineering of the University of Cantabria.

Key words: Automatic Evaluation, Moodle, Programming, PHP

Índice

1. Introducción

1.1. Contexto

Los recientes acontecimientos provocados por la pandemia del COVID-19 han demostrado la importancia de estar preparados para la docencia a distancia, en la cual cobra todavía más relevancia de la que ya tenía la formación en línea o *e-learning*.

En la formación en línea han destacado en los últimos años los Sistemas de Gestión de Aprendizaje (SGA, en inglés, l*earning management system* o LMS). Un Sistema de Gestión de Aprendizaje es una herramienta para crear, distribuir y gestionar distintos tipos de material educativo. Proporcionan recursos para la creación de cursos, entregas, dirección, seguimiento, comunicación y evaluaciones online. Los estudiantes usan el SGA para acceder a recursos, subir tareas, realizar test y compartir información con compañeros y profesores para crear un entorno de aprendizaje dinámico [1].

Hoy en día la mayoría de las instituciones universitarias utilizan herramientas SGA, las cuales están en constante evolución y cada vez otorgan a los profesores más alternativas a la hora de evaluar a los alumnos, como test online o entregas periódicas. Una de las posibles mejoras para los SGA es la automatización de ciertas tareas del profesor. Por ejemplo, en cursos con un gran número de alumnos, supervisar de forma individualizada la evolución de cada estudiante puede llegar a ser muy costoso para el docente. Esto provoca, a su vez, que el seguimiento por parte del profesor no pueda ser tan meticuloso como se desearía. Por ello podría resultar muy útil liberar al profesor de algunas tareas delegándolas en un SGA.

En el caso específico del aprendizaje de un lenguaje de programación, una materia fundamental en cualquiera de los Grados en Ingeniería Informática, resulta de gran importancia la calidad y rapidez de la retroalimentación del docente para que el alumno pueda enmendar sus errores y aprender de ellos. Sin embargo, la corrección manual de los ejercicios propuestos a los alumnos puede ser una tarea ardua y compleja en sistemas de evaluación continua [2].

Otra cuestión de gran relevancia en este tipo de aprendizaje es la detección de plagios. Con bastante frecuencia los alumnos comparten sus soluciones y algunos entregan en estas tareas código ajeno, que posiblemente no comprenden, haciendo pequeñas modificaciones como nombres de variables o similares pensando que así parecerá un código distinto. Resulta fundamental vigilar estas prácticas y que cada alumno desarrolle sus propias soluciones para, en un futuro, saber enfrentarse de manera autónoma a las dificultades que puedan surgir en su actividad profesional.

Uno o varios de estos elementos se podrían automatizar e integrar en los SGA para agilizar la corrección de las soluciones de los alumnos y así mejorar el proceso de aprendizaje continuo de un lenguaje de programación.

Una de las plataformas de SGA más usadas a nivel global es Moodle, de la que hablaremos detalladamente en el apartado 2. Este documento se centra en esta plataforma en el marco de la automatización de procesos de evaluación.

1.2. Trabajo relacionado

Hoy en día el uso de una plataforma SGA como Moodle está totalmente integrado en la mayoría de las universidades. Las universidades pueden necesitar personalizar estas plataformas conforme a sus necesidades o desarrollar proyectos para mejorarlas. En este marco se desarrolla el proyecto EvalCode del grupo de Ingeniería Software y Tiempo Real del Departamento de Ingeniería Informática y Electrónica de la Universidad de Cantabria. Este proyecto consiste en el desarrollo de la herramienta EvalCode, una extensión software de Moodle para la evaluación automática de prácticas de alumnos de programación.

1.2.1 Versiones anteriores de EvalCode

El software EvalCode parte de dos Trabajos de Fin de Grado anteriores [3] [4]. En primer lugar, se inició con un Trabajo titulado “Marco para la evaluación automática de código basado en Moodle”, realizado por Rubén Sebrango. Este TFG implementó la integración de un sistema de evaluación automática de código Java en Moodle. Se desarrolló una versión inicial que tenía funcionalidad, aunque a nivel básico, ya que muchas partes estaban aún en desarrollo. Esto provocaba fallos tanto a nivel de *frontend* como de *backend* que impedían su despliegue y uso.

Posteriormente, el TFG titulado “Mantenimiento y modularización de EvalCode para su puesta en marcha en plataformas Moodle”, de Guillermo Quintana, solventó gran parte de estos fallos en el mantenimiento de la herramienta y realizó un rediseño completo del sistema modularizándolo para facilitar la incorporación de nuevas herramientas en distintos lenguajes de programación. Finalmente se desplegó la aplicación para su uso y se realizaron pruebas en alumnos con las herramientas para lenguaje Java, JUnit y Checkstyle, ya operativas.

De esta manera, este trabajo parte de una versión operativa del sistema de evaluación automática para lenguaje Java, así como un marco genérico de integración de herramientas que abre la posibilidad de integrar otros lenguajes de programación.

Para facilitar la lectura, en el resto del documento nos vamos a referir a la versión del proyecto de Rubén Sebrango como versión “0.1”, a la versión de Guillermo Quintana

como versión “1.0” y la que se ha desarrollado en este proyecto constituirá la versión “2.0”.

1.2.2 Otros trabajos

Además del trabajo previo desarrollado en el software EvalCode, existen gran cantidad de proyectos dedicados a la evaluación automática de ejercicios de programación. Dentro de la plataforma Moodle, también encontramos algunos proyectos con objetivos similares, como, por ejemplo:

- JavaBrat

  Herramienta creada en la *San José State University* en el año 2010 [5]. Consta de dos partes: por un lado, la herramienta Labrat es un software que automatiza la evaluación de programas Java fijando un resultado esperado y viendo si coincide con el de la ejecución del programa (a modo de test); por otro lado, Javabrat, que da nombre a la herramienta, es la parte del programa que da una interfaz web al sistema de evaluación como *plugin* de Moodle. JavaBrat está diseñada para una versión bastante antigua de Moodle y en la actualidad ni siquiera figura en el repositorio de *plugins*, aunque sigue siendo un trabajo de referencia.

- VPL

  Virtual Programming Lab o VPL, es un software complejo y altamente configurable creado en la Universidad de Las Palmas de Gran Canaria. El objetivo del proyecto es proporcionar a los alumnos muchas tareas y apoyar el proceso de gestión y calificación de las mismas. Tiene varios módulos: un *plugin* en Moodle integrado con los módulos de entregas y calificaciones, un editor de código basado en el navegador que permite codificar sin tener que instalar un compilador y un *jail server* que contiene el entorno donde la tarea será evaluada [6]. Soporta una gran cantidad de lenguajes de programación y contiene herramientas para la detección de plagios. Este software, aunque es muy completo, utiliza una interfaz propia para realizar las entregas, ver y modificar el código que está lejos de ser tan visual como la de Moodle y difiere de lo que se busca con EvalCode, una integración total con Moodle siendo su interfaz la misma que la de una tarea original de Moodle, solo que personalizada. Otro aspecto diferencial es que VPL obliga a usar su editor de código *online* como entorno de desarrollo, mientras que EvalCode no interviene más que en la parte de entrega y evaluación permitiendo usar el IDE que se desee en cada asignatura.

- JUnit Exercise Corrector

  El *plugin Moodle JUnit Exercise Corrector (MoJEC)* permite a los estudiantes entregar sus ejercicios Java y que sean probados con una serie de test JUnit

(previamente proporcionados por el profesor) para recibir un *feedback* inmediato con el resultado de los test.

Aunque no hay muchas referencias a este proyecto fuera del repositorio de *plugins* de Moodle y parece en un estado de desarrollo poco avanzado, este trabajo es el más similar a EvalCode en modo de uso. Tiene algunas diferencias a la hora de crear una tarea, pero el sistema de *feedback* en forma de comentario es muy parecido al que proporciona EvalCode.

- CodeRunner

  CodeRunner [7] es un *plugin* de Moodle gratuito, de código abierto y de tipo pregunta, que permite al profesor ejecutar un programa para calificar la respuesta del estudiante a esa pregunta. El uso habitual de CodeRunner se enmarca en cursos de programación donde los estudiantes deben escribir un código y después ese código será calificado con una serie de test. Puede ser utilizado en muchos lenguajes de programación diferentes. Este *plugin*, aun siendo interesante por ser capaz de ejecutar test, está bastante alejado de lo que se busca con el software EvalCode, ya que al ser de tipo pregunta no puede recibir entregas en forma de ficheros.

1.3. Herramientas y lenguajes

A continuación, vamos a ver una recopilación de los lenguajes, herramientas y tecnologías que se han usado para el desarrollo de este proyecto.

1.3.1 Lenguajes

En cuanto a los lenguajes de programación que se han usado en el desarrollo del proyecto, el principal ha sido PHP (*Hypertext Preprocessor*) ya que es el lenguaje en el que está escrito Moodle y en el que se tiene que desarrollar cualquier nueva funcionalidad. También se ha utilizado JavaScript, aunque de manera muy minoritaria. Ha sido necesario porque pequeñas partes en el código de Moodle, como las que manejan *pop-ups*, sí que utilizan *scripts* de código en este lenguaje.

Para realizar pruebas sobre las herramientas también se utilizaron los lenguajes Java y C creando códigos para simular entregas de alumnos. Estos procedimientos se detallarán más a fondo en el apartado de pruebas.

1.3.2 Herramientas y tecnologías para el desarrollo

La herramienta principal en torno a la que gira todo el desarrollo es Moodle. Hablaremos de ella con más detalle en el capítulo 2. Sin embargo, en el transcurso del proyecto también se usaron en mayor o menor medida las siguientes herramientas:

- VirtualBox

  Oracle VM VirtualBox es un software de virtualización multiplataforma que permite extender tu ordenador para ejecutar múltiples sistemas operativos al mismo tiempo [8]. Software clave para el desarrollo de EvalCode ya que para ejecutar y probar Moodle siempre se ha usado una máquina virtual Linux con Moodle instalado.

- Unix Shell

  Este intérprete de comandos inherente a los sistemas operativos Unix será constantemente utilizado durante el desarrollo. Se usará para ejecutar programas para las herramientas además de realizar todo tipo de operaciones.

- SSH y SFTP

  El protocolo SSH, también conocido como *Secure Shell*, es un método para hacer un *login* remoto de un ordenador a otro. Proporciona un sistema de autenticación y asegura la seguridad de la comunicación y la integridad con una fuerte encriptación [9].

  SFTP (*SSH File Transfer Protocol*) es un protocolo para transferir archivos de manera segura. Funciona sobre el protocolo SSH y comparte sus características de seguridad [10]. Estos protocolos fueron de gran ayuda para el acceso gráfico al sistema de archivos de la máquina de manera remota.

- Apache

  El *Apache HTTP Server Project* es un proyecto para desarrollar y mantener un servidor HTTP de código abierto para sistemas operativos modernos. El objetivo de este proyecto es proporcionar un servidor de servicios HTTP seguro, eficiente y extensible en sincronía con los estándares HTTP actuales [11]. Apache Server es el servidor sobre el que se ejecuta Moodle. Para su correcto funcionamiento ha sido necesario modificar la configuración durante el desarrollo teniendo varias veces que cambiar parámetros en función de las necesidades de EvalCode.

- IDEs de desarrollo

  Dadas las características multilenguaje de este proyecto se han usado tres entornos de desarrollo, Visual Studio, Geany y Eclipse:

  - Visual Studio Code es un editor de código fuente ligero pero potente [12]. Es el principal entorno de desarrollo usado a lo largo del proyecto. Con gran cantidad de extensiones para distintos lenguajes, en nuestro caso fueron de muy útiles las de PHP.

  - Geany es un entorno de desarrollo pequeño y ligero. Fue desarrollado para ser rápido y tener pocas dependencias de otros paquetes [13]. Este editor se utilizó para modificar algún archivo de otro tipo como *scripts* o ficheros JavaScript. También fue de gran utilidad su buen funcionamiento

a la hora de hacer búsquedas generales de *strings* en varios ficheros o directorios, en este caso los del *plugin*. Esta última función demostró ser una importante ayuda para moverse por un código de gran densidad como el de Moodle.

    - Por último, Eclipse es un entorno de desarrollo utilizado para diseñar pruebas en lenguaje Java.

- GitHub

  GitHub es una plataforma de desarrollo colaborativo de software que permite alojar proyectos utilizando el sistema de control de versiones Git [14]. Esta plataforma ya se usaba en versiones anteriores de EvalCode y la seguiremos usando continuamente a lo largo del desarrollo. El repositorio se irá actualizando conforme se realicen cambios en la aplicación.

- Meld

  Meld es una herramienta visual para ejecutar los comandos *diff* y *merge* enfocada para desarrolladores. Meld te ayuda a comparar archivos, directorios y distintas versiones de proyectos [15]. Herramienta usada en varias ocasiones a lo largo del proyecto para comparar distintas versiones de EvalCode y ver sus diferencias.

1.3.3 Herramientas ya integradas en EvalCode

Tal como se ha comentado previamente, este proyecto partía de una versión de EvalCode con varias funcionalidades. Estas herramientas se usan para realizar la evaluación automática de programas en lenguaje Java. Sobre estas herramientas ya implementadas en EvalCode se realizarán mantenimiento y actualizaciones. Son las siguientes:

- JUnit

  Es un *framework* Java para escribir test reproducibles. Es una instancia de la arquitectura xUnit [16], dedicada a la creación de pruebas unitarias.

- Checkstyle

  Checkstyle es una herramienta de desarrollo para ayudar a los programadores a escribir código Java que se adhiera a un estándar de codificación. Automatiza el proceso de revisar el código Java para librar a las personas de esta importante pero tediosa tarea [17].

1.3.4 Herramientas nuevas que se integrarán en EvalCode

Además de realizar el necesario mantenimiento del software EvalCode, en este proyecto se integrarán nuevas funcionalidades. Para otorgar nuevas funcionalidades lo que haremos será desarrollar herramientas nuevas en EvalCode. Estas herramientas serán:

- Herramienta de corrección estilo en lenguaje C

  Este software está compuesto de dos partes: Astyle [18], que es un formateador y embellecedor de código fuente; y Style50 [19], específica para lenguaje C, que proporciona una configuración para Astyle y una salida mejorada para la presentación de resultados.

  Es una herramienta de línea de comandos con la que se puede verificar si un código es consistente con la guía de estilo CS50, para lenguaje C. Si el código no cumple estas normas de estilo, la herramienta resumirá los cambios que se deberían hacer al código, resaltando en verde los caracteres que se deberían añadir y en rojo los que se deberían eliminar.

- Firejail

  La herramienta Firejail es un programa de tipo *sandbox* que reduce el riesgo de brechas de seguridad restringiendo el entorno de ejecución de aplicaciones no confiables. El *sandbox* es ligero y con poco *overhead* (carga adicional o sobrecarga) [20]. Todas las características están implementadas directamente en el *kernel* de Linux y accesibles en cualquier ordenador Linux.

- Compare50

  Compare50 es una herramienta de detección de plagio rápida y extensible compatible con varios lenguajes de programación [21]. Proporciona una salida en HTML con las similitudes entre ficheros dispuestas en tablas, así como un ranking de los ficheros que han resultado ser más similares.

1.4. Objetivos

El objetivo principal de este trabajo reside en el mantenimiento, mejora y ampliación del *plugin* EvalCode. Se pretende adaptar el software EvalCode para incluir nuevas funcionalidades, realizar el debido mantenimiento de las herramientas anteriormente implementadas y la puesta en marcha en un servidor de explotación. También se desean realizar mejoras en cuestiones de seguridad y a nivel de interfaz.

Un listado de objetivos más específico sería el siguiente:

- Mantenimiento y actualización de las herramientas ya implementadas. Se realizará una puesta a punto del trabajo realizado en versiones anteriores.
- Integración de herramientas para la evaluación de estilo en programas en lenguaje C. Este tipo de herramienta estaba ya planteada en la anterior versión, pero sin funcionalidad. Por ello, es necesario revisar su integración en el *plugin* para que se ejecute correctamente, procese el resultado y se retorne dentro de la propia plataforma Moodle.

- Integración de un entorno seguro de ejecución de aplicaciones de terceros. Se desea utilizar un software de tipo *sandbox* o virtualización para ejecutar el código de los alumnos de manera segura.
- Integración de una herramienta de detección de plagios. En este apartado, además del sistema de ejecución de la herramienta, habrá que estudiar dónde y cómo implementar una herramienta cuyo resultado está destinado solo al profesor y no a los alumnos. La integración de este tipo de herramientas no existía en versiones anteriores de EvalCode.
- Mejorar aspectos de interfaz de la extensión EvalCode, siendo la visualización del *feedback* uno de los aspectos más relevantes a rediseñar.
- Documentar adecuadamente las nuevas funcionalidades y mejorar los manuales de usuario y las guías para desarrolladores.
- Realizar pruebas reales con alumnos. En este caso se realizarán pruebas de las herramientas para lenguaje Java en la asignatura Métodos de Programación del Grado en Ingeniería Informática de la Universidad de Cantabria. Aquí hay que resaltar que estas herramientas ya están desarrolladas, pero se necesita su mantenimiento, actualización y despliegue a tiempo para ser probadas en la asignatura.
- Desplegar una nueva versión operativa en un servidor de explotación de la Universidad de Cantabria accesible por los alumnos. Esta versión tendría las nuevas herramientas desarrolladas listas para su uso, por lo que marcaría el final del desarrollo.

1.5. Planificación

En este apartado se detalla la planificación del proyecto. Es importante recalcar que se vio afectada por la situación relativa al COVID-19, que provocó un parón total de las actividades de un mes de duración y el posterior cambio a trabajo a distancia, lo cual ocasionó más retrasos. Esto provocó que la estimación inicial de 6 meses de duración de proyecto se viera alargada a unos 8 meses. Como se puede observar en la figura 1, el desarrollo se puede dividir en varias fases:

Una primera de familiarización con la herramienta, donde se deberá leer la documentación disponible de las versiones anteriores y empezar a probar y realizar el mantenimiento de las herramientas ya desarrolladas.

Una segunda fase de desarrollo en la que se realizará la integración de nuevas herramientas. Esta fase realmente se divide en otras 3 fases, una por cada nueva herramienta a implementar:

1. Herramienta para lenguaje C

2. Herramienta *sandbox* de seguridad
3. Herramienta de detección de plagio

Cada una de estas herramientas contarán con sus respectivos periodos previos de preparación y estudio de la herramienta, codificación, posteriores pruebas y documentación, con la creación de guías para desarrolladores y actualizaciones del manual de usuario.

Finalmente, una vez desplegada la herramienta en el servidor de explotación, se realizará una última batería de pruebas general. De esta manera se dará por finalizado el trabajo una vez comprobado que todo funciona correctamente y la herramienta está lista para ser usada con todas las mejoras incorporadas.

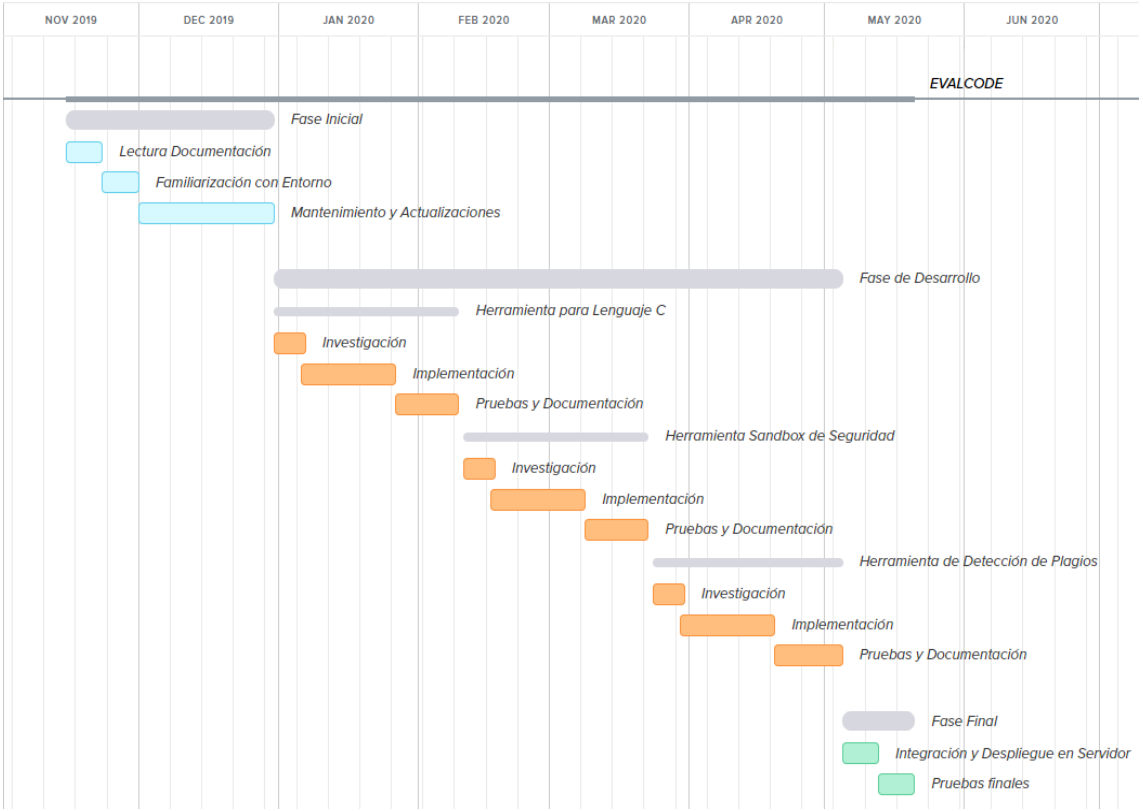


Figura 1: Diagrama de Gantt con detalle de la planificación

2. Moodle

Ahora pasaremos a describir la plataforma Moodle y su funcionamiento, ahondando en las partes que más vamos a tratar en este proyecto. La comprensión de Moodle, especialmente de su sistema de extensiones, es fundamental para el desarrollo de EvalCode.

2.1. Visión general

Moodle es una plataforma de gestión de aprendizaje *online* altamente personalizable que apareció a finales del año 2001. Su nombre es un acrónimo de *Modular Object-Oriented Dynamic Learning Environment* (Entorno de Aprendizaje Dinámico Orientado a Objetos y Modular). Está diseñada para proporcionarle a educadores, administradores y estudiantes un sistema integrado único, robusto y seguro para crear ambientes de aprendizaje personalizados [22]. Es la plataforma de aprendizaje más utilizada del mundo contando, a fecha de marzo de 2020, con más de 190 millones de usuarios registrados repartidos en más de 145.000 sitios web distintos, y estando traducida a más de 120 idiomas.

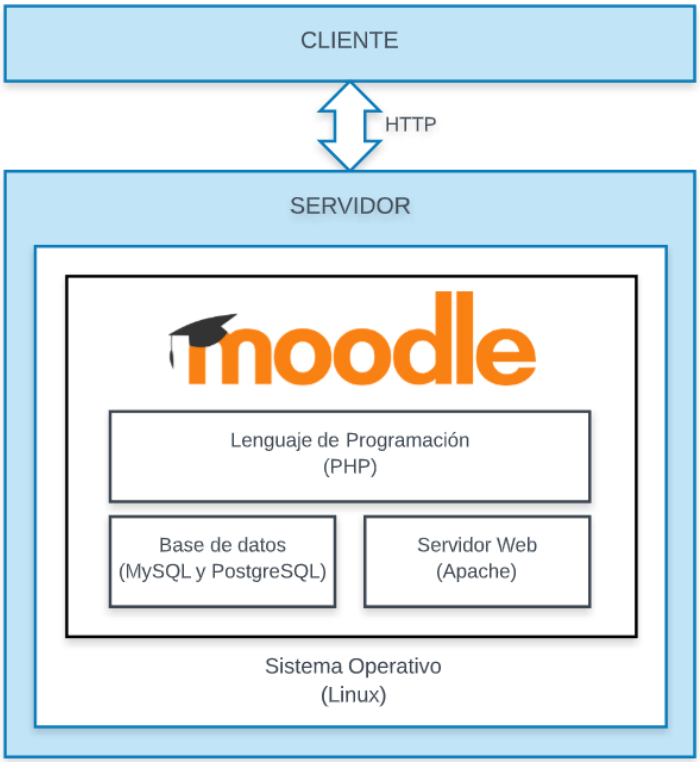


Figura 2: Arquitectura técnica de Moodle

Moodle utiliza la Licencia Pública General GNU, más conocida por sus siglas en inglés GNU GPL, por lo que es un programa *open source* o código abierto. Esto quiere decir que se puede acceder libremente al código fuente y modificarlo para utilizar el

programa desde un servidor web propio. Cualquier institución que instale Moodle está autorizada a su copia, uso y modificación, lo que dota a esta plataforma de una gran comunidad que permite que esté en permanente evolución.

A nivel de arquitectura, como podemos observar en la figura 2, Moodle es una aplicación web que usa de la infraestructura LAMP: **L**inux, como sistema operativo; **A**pache, el servidor HTTP que configura los protocolos de red y establece la conexión entre el servidor local y los clientes del sitio web; **M**ySQL para la gestión de las bases de datos (Moodle también recomienda el uso de PostgreSQL); y por último **P**HP, que es el lenguaje de programación web en el que está desarrollado Moodle.

Centrándose más en el funcionamiento de la aplicación web, Moodle se organiza en cursos, dividendo a los usuarios por roles. Los cursos son creados por un administrador, y al que son añadidos los profesores y alumnos con distintos permisos. Los profesores pueden editar y crear elementos en el curso, mientras que los alumnos solo pueden interactuar con estos elementos. Estos elementos, que pueden ser tareas, entregas, material educativo, test…, son los llamados módulos, de los que hablaremos en el siguiente apartado. En la figura 3 tenemos un ejemplo de la vista principal de un curso.

Figura 3. Ejemplo de la vista principal de un curso Moodle

Es necesario mencionar que, aunque los roles más comunes en Moodle son administrador, profesor y alumno, existen otros: el administrador principal, que es el único que puede crear nuevos administradores; el creador de cursos, que es un profesor autorizado a crear sus propios cursos; y el profesor no editor, que carece de los permisos de edición de curso del profesor, pero que está autorizado a calificar a los alumnos.

2.2. Visión de desarrollo

Como ya se ha explicado, el código fuente de Moodle es de libre acceso y cualquier persona puede descargarlo para intentar personalizar o modificar ciertas partes, pero esto no significa que sea sencillo, ya que puede ser extremadamente denso y complejo para un desarrollador sin experiencia en la aplicación. Para ayudar a los desarrolladores, Moodle otorga una documentación *online* que es de utilidad, pero a veces se antoja insuficiente debido a la gran cantidad de secciones de las que dispone Moodle, sumado a una gran cantidad de APIs, cada cual con un determinado funcionamiento. Otro aspecto que aumenta la complejidad a la hora de documentarse es la gran cantidad de versiones distintas de Moodle, una plataforma que ha sufrido continuas actualizaciones en sus ya casi 20 años de desarrollo.

Moodle tiene un directorio principal de instalación llamado "*moodle/*" en el que se aloja todo el código fuente de la plataforma, que contiene numerosos archivos y subdirectorios. En la instalación de Moodle también se crea el directorio "*moodledata/*". Esta carpeta está ubicada fuera del directorio principal y contiene principalmente archivos de usuarios y archivos de curso. Tiene la particularidad de que es la única carpeta que Moodle habilita con permisos de escritura desde dentro de la aplicación web, por lo que puede ser utilizada para guardar archivos bajados de la plataforma y realizar operaciones con ellos.

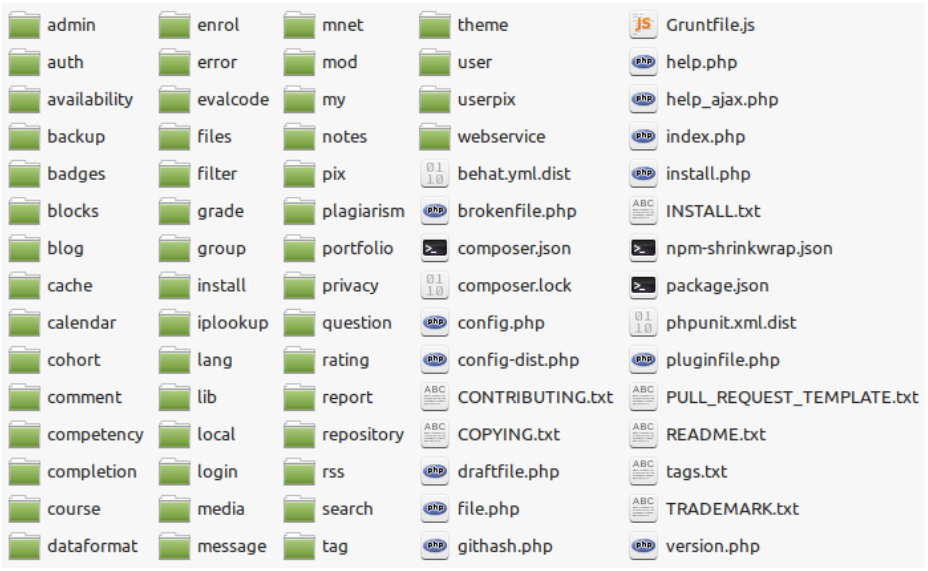


Figura 4: Sistema de archivos de Moodle (contenido de la carpeta *moodle/*)

Aunque Moodle contiene ficheros de diversos tipos, como podemos ver en la figura 4, en su mayor parte estos ficheros son *scripts* en código PHP, los cuales componen el código fuente. En el directorio principal de Moodle se encuentran algunos archivos clave [23], como el fichero *config.php*, que contiene las configuraciones básicas de la plataforma. Este fichero se genera cuando se ejecuta el *script install.php* aunque también puede ser creado y modificado de forma manual. También contiene el archivo

version.php, que es el que define la versión de Moodle a la que corresponde el código y el fichero *index.php*, que contiene la página principal del sitio web.

En cuanto a los subdirectorios que contiene la carpeta "*moodle/*", encontramos algunos como el directorio "*lib/*", que contiene las librerías de funciones del núcleo del sistema; "*admin/*", que proporciona el código para la administración del servidor; "*theme/*", que almacena los temas prediseñados de Moodle y los temas personalizados; o "*blocks/*", que contiene los bloques de información y opciones que se distribuyen en columnas a los lados de las páginas de Moodle. Pero sin duda, el directorio más importante para el desarrollo de este proyecto es el directorio "*mods/*", que contiene los módulos. Los *mods* o módulos son distintas partes de Moodle que aportan una funcionalidad independiente. Cada uno tiene su propio directorio dentro de la carpeta "*mods/*". Los módulos más importantes son los *Activity Modules*, o módulos de actividades, que son los que proporcionan actividades para los cursos. Por defecto hay siete: *Tarea*, *Consulta*, *Foro*, *Glosario*, *Cuestionario*, *Recurso*, y *Encuesta*. Además de estos módulos propios de Moodle, se pueden añadir otros que serán los denominados extensiones o *plugins*, de los que hablaremos en el siguiente apartado.

Por otra parte, existen otros elementos que también son de gran importancia para el desarrollo de un *plugin* en Moodle. A continuación, se describen brevemente los más relevantes para el desarrollo del proyecto.

2.2.1 APIs

Una API o Interfaz de Programación de Aplicaciones, en inglés *Application Programming Interface*, son un conjunto de funciones, subrutinas y protocolos diseñados para desarrollar e integrar software en aplicaciones. Moodle contiene en su *core* o núcleo más de 40 APIs distintas que son las que proporcionan herramientas para desarrollar los *scripts*, por lo que son esenciales a la hora de crear un *plugin*. Aunque se han utilizado varias de estas APIs, la File API, String API y Form API han sido especialmente importantes durante el desarrollo.

- File API

  La File API es de gran importancia en el proyecto, ya que es la que maneja todos los archivos que se almacenan en cualquier parte de la plataforma Moodle. Esta API no se usa para operar con archivos internos de Moodle, sino que es la que permite almacenar o descargar archivos del sitio web. Estos archivos se almacenan en una base de datos central, solo accesible por la File API, donde cada archivo está asociado a una *file area*. Estas *file areas* son espacios conceptuales para identificar los ficheros. Las *file areas* tienen múltiples características como contexto, curso o tipo para ser identificadas, con el objetivo de que cada fichero esté correctamente asociado a su curso, tipo de recurso, tarea, usuario…

El manejo y procesamiento de ficheros es clave al crear una tarea del tipo EvalCode y al ejecutar cualquier herramienta de evaluación automática.

- String API

La String API es la que maneja los diccionarios de la aplicación, siendo la que obtiene los *strings* definidos dentro de estos diccionarios para usarlos en la interfaz gráfica. Estos contenedores de *strings* se almacenan en el directorio "*lang/*" de Moodle, aunque también cada módulo puede tener una carpeta "*lang/*" con su diccionario específico. Esto es especialmente interesante en el caso de los *plugins*, ya que da facilidades a la hora de la instalación, llevando ya incorporados los *strings* extra que necesite. La API también se ocupa de la internacionalización, pudiendo tener dentro de los directorios "*lang/*" diccionarios en varios idiomas. En la figura 5 podemos ver un ejemplo de la definición de *strings* en un diccionario.

```
$string['nombreplugin'] = 'Este es el nombre de mi plugin'
$string['descripcionplugin'] = 'Este plugin es un ejemplo'
```

Figura 5: Definición de un *string* en un archivo de diccionario

La obtención de estos *strings* se realiza con la función "*get_string*()", usada constantemente en todo el desarrollo. Esta función puede recibir varios parámetros, pero entre ellos no pueden faltar el identificador del *string* y el identificador del archivo de diccionario en el que tiene que buscarlo. En la figura 6 podemos ver la obtención de uno de estos *strings* con la llamada a "*get_string*()".

```
echo get_string('nombreplugin', 'modulo_nombreficherolangdelplugin')
```

Figura 6: Uso de la String API para obtener un *string* de un diccionario

- Form API

La Form API es utilizada por Moodle para crear todos los formularios web. Soporta todos los elementos propios de HTML adaptados a la plataforma con características mejoradas de accesibilidad y seguridad. Con más de 30 formularios distintos Moodle proporciona muchos mecanismos para añadir elementos de este tipo a la interfaz cuando se desarrolla un *plugin*, aunque puede ser bastante tedioso aprender a usarlos ya que cada uno tiene un funcionamiento distinto. De los formularios y del uso que les hemos dado en el proyecto se hablará con más detalle en los siguientes apartados.

2.2.2 Renderers

Un *renderer* [24] es un *script* PHP que maneja todas las salidas de cualquier componente de Moodle. Todos los componentes de Moodle han de tener sus *renderers* propios. El objetivo de usar *renderers* es permitir a los temas tener completo control sobre el HTML que se genera en cada página de Moodle. De este modo los *renderers* fijan un tema HTML con un formato que no se puede modificar.

El *renderer* no debe contener lógica más allá de lo necesario para generar la salida de la página. Esta salida dependerá de cada componente. Por ejemplo, el foro tendrá un método para mostrar un post, para mostrar un hilo y para hacer un formulario de búsqueda.

Estos *renderers* limitan la salida HTML que se puede mostrar en Moodle y, por tanto, serán claves para el formato de salida de los comentarios de *feedback* generados por las herramientas en EvalCode, tal y como se comentará en el apartado 3.2.3.

2.3. Extensiones

Moodle es respaldado por una fuerte comunidad que desarrolla multitud de *plugins*. Las extensiones o *plugins* son módulos nuevos que se añaden a Moodle para darle nuevas propiedades o funcionalidades. En este caso nos vamos a centrar en las extensiones de tipo *Activity Module*, entre las que se encuentra EvalCode. Este tipo de módulos contienen una serie de elementos fijos que son comunes a todos ellos, a los que se les suman los *scripts* específicos para cada módulo. Como podemos observar en la figura 7, los elementos que todo *Activity Module* ha de tener son [25]:

- **mod_form.php**: se usa cuando se añade el módulo a un curso. Contiene los elementos que se mostrarán en el formulario de creación o instalación del módulo.
- **version.php**: contiene una serie de propiedades que se usan durante la instalación o actualización del *plugin*. Permite verificar que el *plugin* es compatible con la versión de Moodle que se está utilizando.
- **icon.gif**: icono que representa el módulo, en el caso de EvalCode se usará el mismo icono que el de la tarea original de Moodle.
- **db/install.xml**: define la estructura de las tablas para todas las bases de datos. Se usa en la instalación del módulo.
- **db/upgrade.php**: define los cambios en la estructura de las tablas. Se utiliza cuando se actualiza el módulo.
- **db/access.php**: define los permisos de acceso que tendrá cada parte del módulo.
- **index.php**: esta página se utiliza para presentar la lista de todas las instancias del módulo en un determinado curso, es decir, todas las actividades de este tipo que se han creado en un curso.

- **view.php**: este *script* proporciona los enlaces a la vista cuando una página genera su *layout* y actividades.
- **lib.php**: este fichero contiene todas las funciones definidas para el módulo, incluyendo algunas comunes y otras propias de cada módulo.
- Por último, como ya se explicó en el apartado 2.2.1, en cada módulo habrá una carpeta **lang/** con archivos de idioma o diccionarios que contienen *strings* específicos para ese módulo.

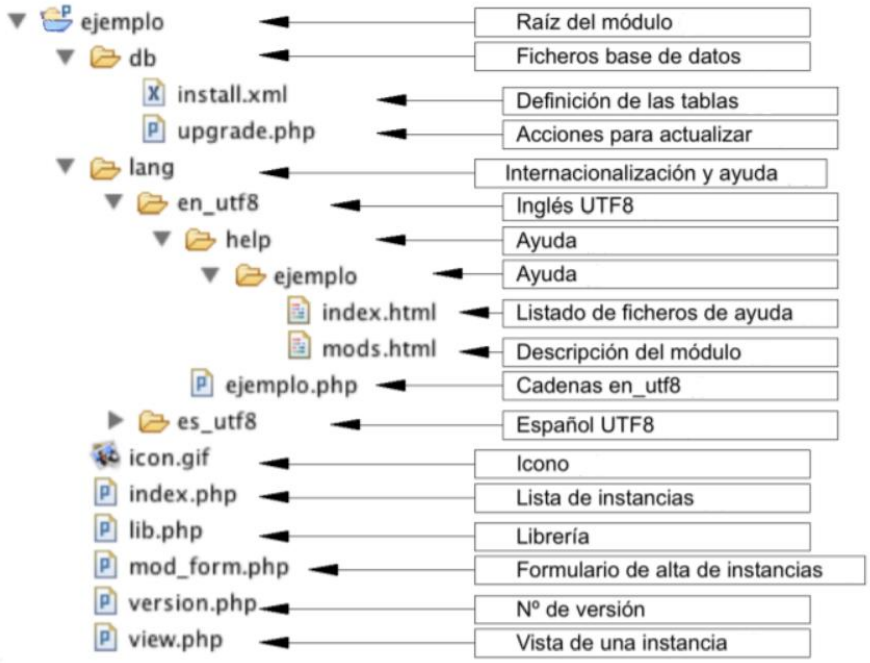


Figura 7: Sistema de ficheros básico de un *Activity Module*. Fuente: [26]

Este es el conjunto mínimo de ficheros que necesita un módulo. Para añadir nuevas funcionalidades y/o personalizar el módulo, es necesario añadir nuevos *scripts* o modificar los existentes.

2.3.1 Formularios

Como ya se ha comentado en el apartado 2.2.1, los formularios son la parte de los módulos dedicada a crear los formularios web. Una parte fundamental a la hora de crear un *plugin* consiste en la posibilidad de añadir formularios en cualquiera de las vistas del módulo. En la figura 8 tenemos un ejemplo de formulario, en este caso, uno correspondiente a la creación de una actividad del módulo *Tarea*.

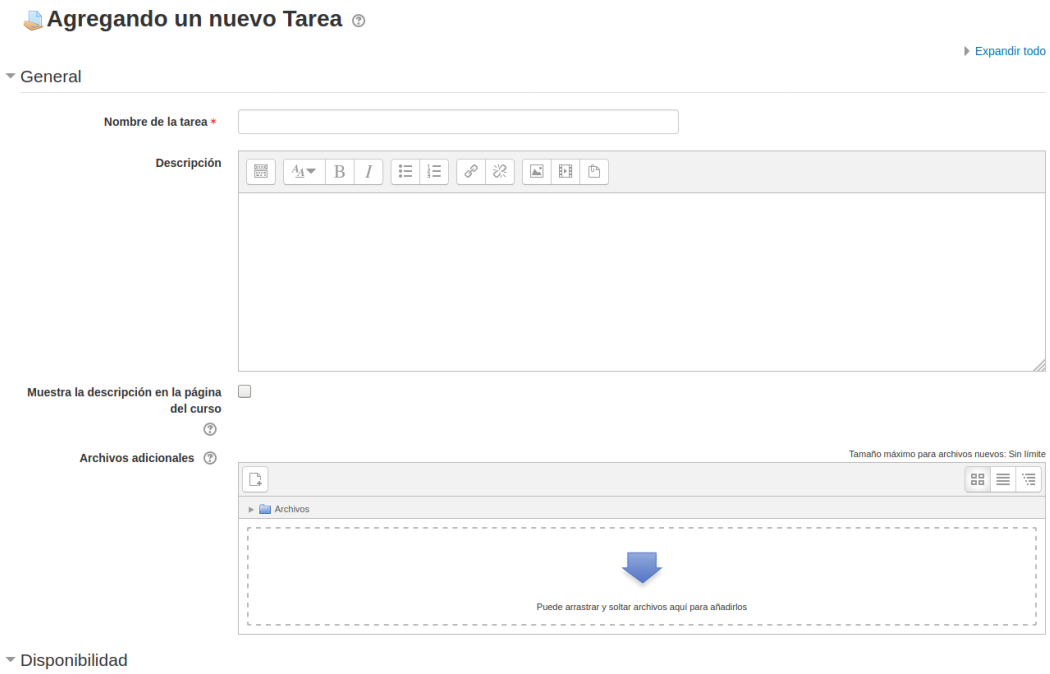


Figura 8: Ejemplo de formulario de creación de una tarea básica en Moodle

Para añadir un nuevo formulario, el procedimiento habitual es crear un nuevo *script* PHP (cuyo nombre de fichero habitualmente termina en *form*, de acuerdo a la guía de estilo de Moodle) y que tiene una estructura muy determinada. La clase que se crea en el formulario ha de heredar la clase *moodleform* que está definida en el archivo "*formslib.php*", que necesita ser importado y que se encuentra en el directorio de librerías de Moodle. El método donde se define el formulario y se añaden elementos es el método "*definition*()", que hace la función de constructor y que no recibe parámetros. En este método es imprescindible tener una instancia del formulario en la variable "$*mform*" para añadir elementos de formulario con sus parámetros y opciones.

Tras haber añadido elementos al formulario en el método "*definition*()", se pueden añadir otras funciones como "*validation*()", que permite comprobar la validez de los datos que se han añadido al formulario o si se han rellenado todos los campos obligatorios. El método recibe dos parámetros, "$*data*" y "$*files*": el primero contiene en un array los datos añadidos a cada uno de los campos y el segundo los ficheros adjuntados. Si se detecta un error, el procedimiento habitual es retornar un array de errores; en caso de que todo sea válido, se retorna el valor *null*. En la figura 9 podemos ver un ejemplo del código de creación de un formulario, incluyendo los métodos de definición y validación [27].

```
//moodleform está definido en formslib.php
require_once("$CFG->libdir/formslib.php");

class simplehtml_form extends moodleform {
    //Añadir elementos al formulario
    public function definition() {
        global $CFG;
        $mform = $this->_form; // No olvidar la _

        // Añadiendo elementos al formulario
        $mform->addElement('text', 'email', get_string('email'));
        //Fijando el tipo de elemento
        $mform->setType('email', PARAM_NOTAGS);
        //Fijando el valor por defecto
        $mform->setDefault('email', 'Please enter email');
        ...
    }

     //Aquí se debe añadir la validación personalizada
     function validation($data, $files) {
        $errors = parent::validation($data, $files);
        if($data['ejemplo'] < 0) {
            $errors['ejemplo'] = 'Error, el dato debe ser mayor que 0.';
        }
        …
        return $errors;
    }
}
```

Figura 9: Ejemplo del código de definición de un formulario

Una vez creado el formulario se podrá instanciar desde un *script* externo, en uno de los métodos que maneja la vista. En la figura 10 se puede ver un pequeño ejemplo de cómo crear una instancia de ese formulario.

```
//incluir simplehtml_form.php
require_once('PATH_TO/simplehtml_form.php');

//Crear instancia del formulario
$mform = new simplehtml_form();
```

Figura 10: Ejemplo del código de creación de una instancia de un formulario

Como acabamos de ver, a los formularios hay que ir añadiéndoles elementos con distintas funciones. El elemento de formulario más relevante y sobre el que más se ha trabajado en nuestro desarrollo es el *filemanager*, que se incluye en la File API. El *filemanager* habilita un espacio para adjuntar archivos en la plataforma. En la figura 11

podemos ver un ejemplo de cómo se ve en una página de Moodle. Hay algunos elementos que hacen funciones similares como el *filepicker*, pero este tiene algunas desventajas, como que no se puede adjuntar más de un fichero. Estos ficheros que se suben a Moodle se almacenan internamente por si es necesario utilizarlos de nuevo.

Figura 11: Ejemplo de vista de un formulario de tipo *filemanager*

Otros elementos de formulario relevantes en el proyecto son *select*, que te permite crear un sencillo menú desplegable para elegir entre varias opciones, y *submit*, que crea un botón para pulsar y ejecutar la acción asociada. En el proyecto EvalCode, el botón *submit* ejecuta la acción seleccionada a través del menú *select*. En el apartado 3.3.4 se explicará más detalladamente el uso que se les ha dado a estos elementos de formulario.

3. La extensión EvalCode

3.1. Introducción

En este apartado nos centraremos en la extensión sobre la que se ha trabajado en este proyecto. El sistema de automatización de procesos de evaluación EvalCode es una extensión de Moodle de tipo *Activity Module* basado en la estructura del módulo *Tarea*.

El funcionamiento general del sistema es el siguiente: una tarea de tipo EvalCode, una vez creada y configurada, permite a los alumnos realizar entregas adjuntando ficheros. Estos archivos son descargados y guardados en el sistema de almacenamiento de Moodle dentro la carpeta "*moodledata*/". A continuación, mediante una función PHP, se realizan una serie de comandos de *shell* para ejecutar las herramientas (por ejemplo, de corrección de estilo) que trabajan sobre estos ficheros. Por último, el *plugin* procesa el resultado de esas herramientas y se lo muestra al usuario mediante el sistema de retroalimentación de la tarea. Este proceso se puede apreciar de manera gráfica en la figura 12.

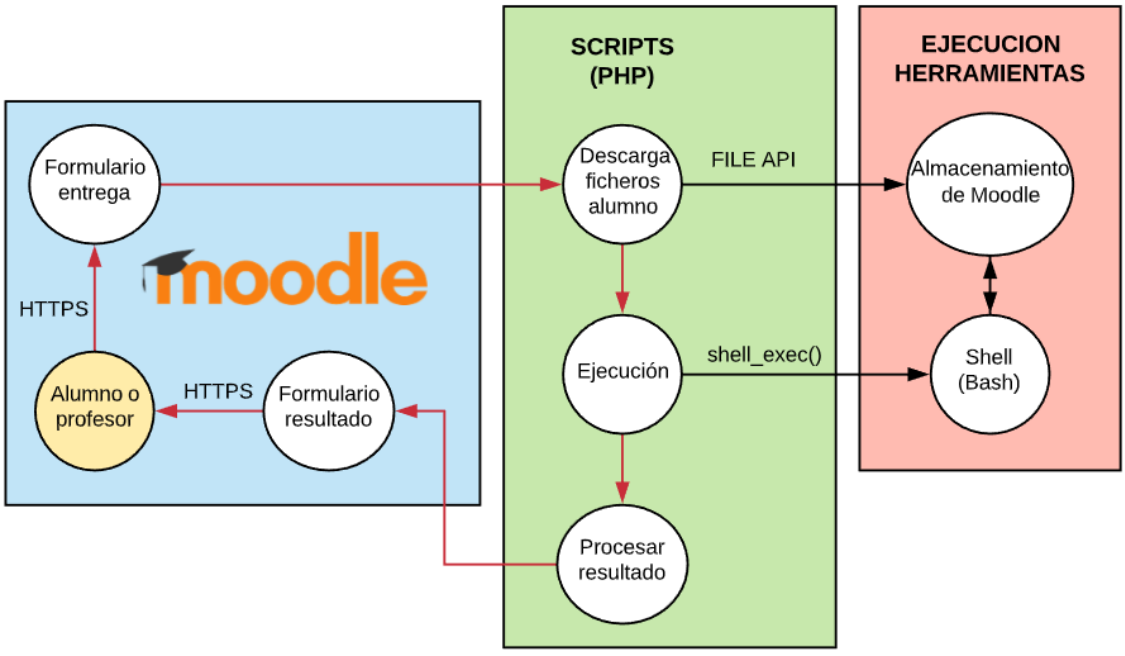


Figura 12: Diagrama con el funcionamiento de EvalCode

3.1.1 Estado previo

Tal y como se ha comentado en la sección 1.2.1, este proyecto parte de la versión "1.0" del software EvalCode, una versión que se encontraba en un estado estable y propicio para el desarrollo de nuevas herramientas, aunque todavía con algunos fallos en el código que se detallarán en el apartado 3.2.

En la “v0.1” del software se creó el módulo con una funcionalidad básica y se integraron dos herramientas para lenguaje Java: JUnit y Checkstyle, de las que ya hemos hablado brevemente. Estas dos herramientas eran ejecutadas por EvalCode para obtener una nota y un *feedback* en forma de comentario. Esta versión “0.1” todavía no disponía de la funcionalidad necesaria para ser desplegada en el servidor y probada con alumnos, pero fue la que sentó las bases del proyecto.

Uno de los objetivos principales de la “v1.0” fue solventar gran parte de los fallos de la “v0.1” para que se pudiera desplegar en un servidor de explotación. Se consiguió que las herramientas para lenguaje Java fueran totalmente funcionales y probadas con alumnos. Además, esta versión “1.0” incorporó la modularización de las herramientas de evaluación. En este contexto, se entiende por modularización la separación de las herramientas de evaluación del resto de los *scripts* y a la automatización del proceso de detección de una nueva herramienta por parte del módulo. De forma general, esta característica facilita a los desarrolladores la incorporación de nuevas herramientas.

Para que incorporar nuevas herramientas sea un proceso estandarizado, el código de inclusión y ejecución de las herramientas en el *plugin* EvalCode se movió a un directorio llamado “*tools*/”, que es donde se organiza esta nueva estructura modular. Este cambio es de gran importancia en nuestro desarrollo, ya que se utilizará constantemente este sistema modular, bien para realizar mantenimiento sobre las herramientas ya implementadas o para incorporar otras nuevas.

3.1.2 Configuración del servidor Moodle

Para desarrollar un *plugin* en Moodle, además de familiarizarte con lo que es estrictamente el código fuente, es importante configurar debidamente el entorno en el que se va desarrollar. Como ya hemos comentado, Moodle es software libre y cualquier persona puede descargar el código fuente completo y crear un entorno aislado donde probar Moodle y realizar el desarrollo de un nuevo *plugin* sin restricciones. En el proyecto EvalCode, desde la versión “0.1” se trabaja sobre una máquina virtual Linux ejecutada sobre el software VirtualBox. Esta máquina virtual fue proporcionada por el Centro de Formación en Nuevas Tecnologías (CeFoNT) de la Universidad de Cantabria. Para la configuración del servidor Moodle de EvalCode en la máquina virtual se deben llevar a cabo los siguientes procesos:

- **Instalación de Moodle**
  La instalación de Moodle requiere tener ya instalado el Apache Server, que será el servidor que nos permitirá conectarnos con Moodle, así como las herramientas para el desarrollo con PHP. Actualmente Moodle se instala en la carpeta “*var/www/html*” que es donde se alojan las páginas y aplicaciones web de Apache. En ese directorio se encuentra el directorio principal de Moodle, “*moodle*/”. En la instalación se debe de crear también la carpeta “*moodledata*/”, de la que se ha hablado en el apartado 2.2 y que se sitúa fuera del directorio del

servidor, en este caso en "*var/www*". Finalmente, la instalación requiere configurar distintos parámetros del fichero "*config.php*", que se encuentra en el directorio principal de Moodle, como, por ejemplo, la dirección al directorio de archivos de Moodle, "*var/www/moodledata*" o la dirección web de salida para Moodle, en este caso "https://moodle.ctr.unican.es".

- **Configuración de red y acceso remoto**
  Otro aspecto relevante es la configuración de los servicios de red. En el Anexo (apartado 7) detallaremos los pasos a realizar para configurar la red en el servidor de pruebas utilizado para el desarrollo local.

  Esta configuración permite acceder a la web de Moodle desde el navegador, a través de la dirección "https://moodle.ctr.unican.es". Asimismo, permite acceder al sistema de archivos de la máquina virtual de manera gráfica mediante el protocolo SFTP. Esto facilita en gran medida el desarrollo, ya que la máquina virtual Linux utilizada no cuenta con interfaz gráfica.

- **Instalación del plugin**
  La instalación de nuevos *plugins* está automatizada en Moodle. Para que Moodle detecte el *plugin* solamente se ha de mover la carpeta del módulo EvalCode, llamada "*evalcode/*", dentro del directorio "*mod/*" de Moodle. Además, para que el *plugin* funcione correctamente, se requiere la instalación de las herramientas de evaluación integradas.

3.1.3 Visión de desarrollo de EvalCode

EvalCode contiene todos los elementos comunes a los *Activity Modules*, descritos en el apartado 2.3, pero además utiliza otros *scripts* que añaden nuevas funcionalidades. En la figura 13 podemos apreciar el contenido del módulo EvalCode en su versión "2.0".

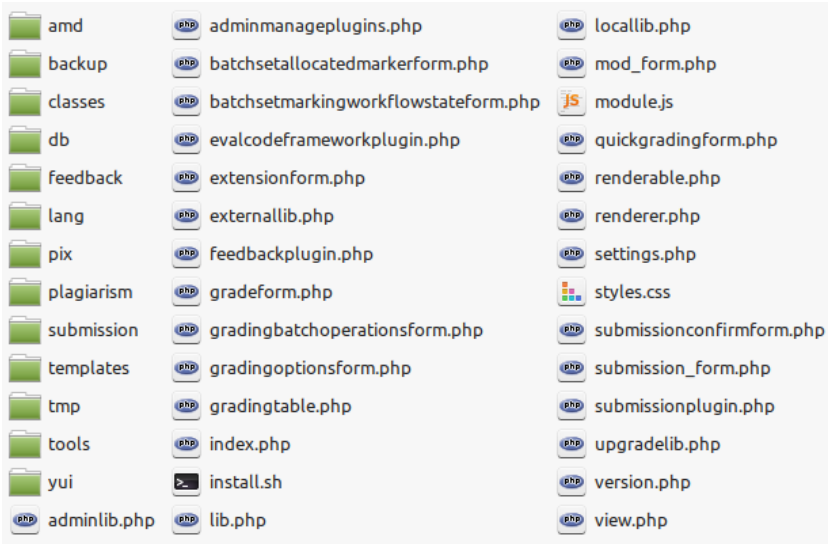


Figura 13: Sistema de archivos de EvalCode

La estructura del módulo está basada en un diseño de tres capas: la capa de datos, en la que se encuentra el servidor de la base de datos; la capa lógica, dónde se encuentran los *scripts* que aportan funcionalidades a EvalCode; y la capa de presentación, que corresponde a los formularios necesarios para generar las páginas del *plugin*.

A continuación, se repasarán los elementos de la extensión que más se han utilizado específicamente en el desarrollo de la versión "2.0" de EvalCode:

- Directorio "*tools*/"

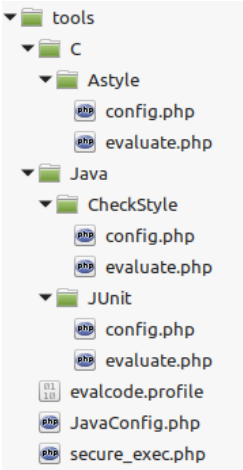


Figura 14: Estructura de ficheros de la carpeta *tools*/

El directorio "*tools*/" se creó en la versión "1.0" para facilitar la modularización de las herramientas. Está basado en una estructura de directorios, como se puede apreciar en la figura 14. En un primer nivel se clasifican las herramientas en directorios por lenguajes de programación, y en un segundo encontramos ya las carpetas que contienen los *scripts* específicos de cada herramienta. Cada herramienta tiene que contener, por lo menos, dos archivos: "*config.php*" y "*evaluate.php*".

- *config.php*

  El *script* "*config.php*" se utiliza al crear una actividad del módulo EvalCode. Contiene la configuración para cada herramienta de EvalCode, siendo aquí donde se define la información acerca de la herramienta que se mostrará al usuario a la hora de elegir entre las herramientas disponibles en el *plugin*.

- *evaluate.php*

  El *script* "*evaluate.php*" contiene el código de ejecución y el de procesamiento del resultado de cada herramienta. De manera común para

todas las herramientas, este fichero contiene una única función, llamada $*evaluatefunc*, y que recibe tres parámetros: $*path*, con el directorio donde se encuentran los ficheros de la entrega del alumno, siempre dentro de "*moodledata/*"; $*returndata* variable que servirá para devolver el resultado; y $*additionalParams*, variable que contiene parámetros extraídos del formulario de creación de la actividad (estos parámetros son opcionales y pueden ser usados para obtener la calificación o para usar en la ejecución de la herramienta).

La función $*evaluatefunc* es invocada después de la entrega de una tarea EvalCode en la que se haya seleccionado esa herramienta. La llamada a esta función se realiza desde el método "*process_save_submissions*()", en el *script* "*locallib.php*", del que hablaremos a continuación.

- locallib.php

  El fichero "*locallib.php*" contiene la mayor parte del código que aporta la funcionalidad específica de EvalCode. Entre otras, las siguientes funciones son las más relevantes:

  - *process_save_submissions*()

    Esta función es la que se invoca cuando el alumno pulsa sobre el botón de guardar entrega. La función guarda la entrega del alumno, llama a los *scripts* "*evaluate.php*" de las herramientas seleccionadas por el profesor y muestra al alumno el resultado que devuelven estos *scripts* como una evaluación de su tarea, con una nota y con su correspondiente *feedback* que explique más detalladamente la calificación.

  - *process_grading_batch_operation()*

    Esta función es la encargada de administrar las operaciones que se pueden llevar a cabo por parte del profesor en la vista de todas las entregas de los alumnos. A esta vista se accede desde el rol de administrador, profesor o similares, pulsando sobre una tarea y entrando con el botón "View all submmisions", que significa ver todas las entregas.

- Formularios

  En este documento ya han sido ampliamente descritos los formularios, tanto su API como su creación y funcionamiento. Ahora nos vamos a centrar específicamente en los que han sido más importantes en nuestro desarrollo:

- *mod_form.php*

  Una de las partes más importantes del *plugin* es su página de creación de una nueva actividad, generada a través del formulario "*mod_form.php*". Este formulario es el encargado de recoger toda la configuración de la tarea y permite seleccionar los parámetros necesarios para la creación de una tarea EvalCode. Además de los parámetros configurables propios del módulo *Tarea*, una tarea EvalCode permite elegir qué herramientas de entre todas las integradas se van a utilizar en la evaluación, así como el peso que van a tener cada una de las herramientas en la nota final.

  Asimismo, existen dos elementos de formulario del tipo *filemanager* que permiten que el profesor adjunte ficheros. El primero de estos *filemanager* sirve para adjuntar archivos que se desee proporcionar al alumno para realizar la actividad. El segundo, que es específico de EvalCode, está destinado a que el profesor suba los llamados ficheros de evaluación. La función de estos ficheros es sustituir algunos archivos que forman parte de la entrega de los alumnos, para que estos no puedan modificarlos. Este aspecto es relevante ya que, por ejemplo, los ficheros de test JUnit podrían ser modificados para manipular los test unitarios y conseguir así que JUnit los evalúe como correctos sin haber realizado adecuadamente la tarea que el profesor planteaba al alumno. Para que la herramienta funcione correctamente el archivo adjunto ha de ser un fichero ".zip" con la misma estructura del que entregará el alumno. El resto de elementos del formulario "*mod_form.php*" son idénticos al módulo *Tarea* original de Moodle.

- *gradingbatchoperationsform.php*

  Este formulario está dedicado específicamente a añadir un desplegable de opciones y un botón de confirmación. Lo podemos ver en la figura 15, siendo estos elementos un *select* (izquierda) y un *submit* (derecha).

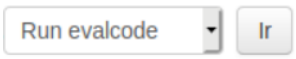


Figura 15: Vista del formulario definido *gradingbatchoperationsform.php*

3.2. Actualizaciones y actividades de mantenimiento

En este apartado nos vamos a centrar en el mantenimiento y actualizaciones realizados en el proyecto. El mantenimiento se puede clasificar en cuatro tipos [28]: correctivo, adaptativo, perfectivo y preventivo. El mantenimiento correctivo tiene por objetivo localizar y eliminar los posibles defectos de los programas, que pueden darse incluso ya finalizada la etapa de pruebas. El adaptativo realiza modificaciones en los programas

para responder a cambios en el entorno (hardware o software) en el que se ejecuta el sistema. El mantenimiento perfectivo es la incorporación de nuevas funcionalidades por cambios en los requisitos de un software. Por último, el preventivo tiene como objetivo mejorar el software para evitar la aparición de fallos o mitigar sus consecuencias.

El mantenimiento y actualizaciones ha constituido una de las actividades más costosas durante el desarrollo de EvalCode. Se han encontrado varios problemas o elementos susceptibles de mejora, que se resumen en la tabla 1.

Defecto Encontrado	**Mantenimiento**	**Solución resumida**
Error cálculo de la nota de JUnit, nunca suspende	Correctivo	Solucionar el fallo en la función de cálculo de la nota (3.2.1)
El cálculo de la nota de JUnit y Checkstyle no es el adecuado	Perfectivo	Rediseño de las fórmulas de cálculo de la calificación de las herramientas JUnit y Checkstyle (3.2.1)
Nombre de la carpeta que contiene las herramientas poco intuitivo	Preventivo	Cambiado de "*languageConfig*" a "*tools*", actualizadas referencias al directorio en el *script* "*locallib.php*"
Al ejecutar EvalCode aparece una pantalla de *warnings* o errores leves	Correctivo	Se eliminan los errores uno por uno modificando el código (3.2.1)
Error al hacer una entrega vacía	Correctivo	Cambio en el código de "*localib.php*" para dar un aviso si se realiza una entrega sin archivos adjuntos (3.2.1)
EvalCode solo admite entregas de archivos .zip	Perfectivo	Añadida posibilidad de entregar ficheros no comprimidos (3.2.1)
Error de Checkstyle está dando siempre nota 100	Correctivo	Cambio dirección errónea de archivo necesario para la ejecución (3.2.1)
No se escribe la lista de sucesos, lo que dificulta la depuración de errores	Correctivo	Cambios en el fichero de configuración de Apache y uso de nueva función (3.2.1)
Necesaria actualización de JUnit4 a JUnit5	Adaptativo	Cambio en la ejecución y procesamiento de toda la herramienta JUnit (3.2.2)
El *feedback* en HTML del Style50 no se muestra de manera adecuada	Adaptativo	Refactorización de la herramienta para generalizar la salida de las herramientas del profesor (3.2.3)

Tabla 1: Actividades de mantenimiento

3.2.1 Operaciones menores de mantenimiento

Las dos principales acciones de mantenimiento que se llevaron a cabo fueron la actualización de JUnit y el cambio en el sistema de *feedback* de EvalCode, pero también se solventaron otros defectos menores, como los siguientes:

- **Cambios en la obtención de las notas**
  El sistema de cálculo de la calificación que JUnit daba al alumno era erróneo. En la función $*evaluatefunc* de la herramienta JUnit, la parte dedicada a este cálculo estaba programada de tal forma que el alumno nunca suspendía. Por ello se corrigió la fórmula para que si el alumno tuviera menos test correctos de los indicados en el parámetro que se proporciona al crear la actividad, su nota fuera menor de 5.

  Tras revisar los sistemas de cálculo de la calificación en Checkstyle también se detectó que este cálculo, que se hacía en función del parámetro de máximo número de errores permitidos, no suspendía si se superaban ese número de errores. Debido a esto se cambió la fórmula para que, si se superaba ese número de errores o *warnings*, la calificación fuera 0.

- **Pantalla de errores al ejecutar EvalCode**
  En la versión "1.0" aparecía una pantalla de aviso de errores que no afectaba a la ejecución, pero que provocaban que la ejecución no fuera totalmente fluida. La mayor parte de estos errores se solucionaron fácilmente con pequeños cambios en el código fuente. Uno de ellos estaba relacionado con el sistema de envío de *emails* de notificación a los alumnos una vez habían realizado una entrega, que fue desactivado, ya que no se va a utilizar en EvalCode.

- **Error de entrega vacía**
  Se detectó un error cuando un alumno realiza una entrega vacía, que provocaba un fallo inesperado en la plataforma. Este error se corrigió incorporando en el método que maneja las entregas una simple estructura condicional para que, en caso de que hubiera una entrega vacía, se notificase y se recargase la página.

- **No se permiten entregas de archivos sin comprimir**
  Para permitir entregas de archivos no comprimidos se modificó el código de la función "*process_save_submmision*()" para que si el archivo entregado en la tarea no es de tipo ".zip", no se llame a la función de descompresión si no que directamente se descargue en la carpeta temporal de evaluación.

- **Error Checkstyle**
  Otro defecto que se detectó fue que Checkstyle no estaba funcionando correctamente ya que siempre otorgaba la máxima calificación. Al depurar el fallo se vio que no se estaba apuntando bien a la ubicación de un archivo necesario para la ejecución de la herramienta. Este fallo se debe a una

discordancia entre versiones del *plugin*, ya que en anteriores versiones cambiaba la organización del sistema de ficheros.

- **No se escribe la lista de sucesos en la ejecución de EvalCode**
  La lista de sucesos o *logs*, como se les suele llamar en inglés, se escriben en archivos de texto que se van rellenando con información mientras se realiza la ejecución de un software. Estos *logs* pueden ser de mucha ayuda para depurar errores. En EvalCode se detectó que no había ninguna manera de visualizarlos ya que no se podía escribir en ningún fichero por problemas de configuración. Para habilitar la escritura de estos *logs*, se cambiaron la configuración de Moodle para habilitar ciertas características de su modo de depuración (por ejemplo, se activaron los mensajes en modo desarrollador y se habilitó la opción *display debug messages*). La documentación de Moodle contiene algunos tutoriales a este respecto [29].

  También se realizaron cambios en el fichero de configuración de Apache "*/etc/apache2/7.1/php.ini*", donde se habilitaron las opciones de escritura de *logs* del sistema (esto es, la visibilidad y la ubicación de los ficheros).

  Para escribir los logs se utiliza la función PHP "*error_log*()". En la figura 16 tenemos un ejemplo de uso de esta función:

```
error_log (string $message, int $message_type = 3, string $destination)
```

Figura 16: Definición de la función "*error_log*()"

3.2.2 Actualización de JUnit

Para las primeras pruebas en la herramienta se utilizaron prácticas de asignaturas de años anteriores que utilizaban la librería JUnit en su versión 4, en el entorno de desarrollo Eclipse, para ejecutar test unitarios sobre el código de los alumnos. Estas pruebas fueron satisfactorias ya que todo funcionaba de manera correcta. Sin embargo, al diseñar las pruebas en alumnos, de las que hablaremos en el apartado 4.2, el resultado de la ejecución de JUnit no era el esperado. Esto era debido a que los docentes habían actualizado la librería que se usaba para hacer test unitarios en sus tareas en lenguaje Java, usando ahora nueva versión de JUnit, la versión 5 o *Jupiter*.

La herramienta JUnit no funcionaba cuando se usa esta nueva versión porque cambian las librerías que se deben importar y ciertos aspectos a la hora de diseñar los test, como algunos parámetros que reciben las funciones. Además, esta nueva versión de JUnit también ha rediseñado el método de ejecución y salida por consola, por lo que hubo que cambiar todo el código ejecución y procesamiento de la herramienta que se realizaba en el fichero "*evaluate.php*" de la herramienta JUnit.

En la figura 17 se puede ver el fragmento de la función $*evaluatefunc* que fue rediseñado para esta actualización. En esta función se lleva a cabo: la compilación del código Java entregado por el alumno; la ejecución la herramienta JUnit 5; la conversión de la salida de la herramienta a HTML con el comando "*ansi2html*"; y el procesamiento de esta salida donde se extraen el número de test totales, correctos y fallados, con los que se realiza el cálculo de la calificación. El *feedback* que se retorna será la salida por consola en formato HTML.

```php
$evaluatefunc = function ($path,$returndata,$additionalParams){

    shell_exec('find * -name "*.java" > sources_list.txt');

    //En la compilacion se añaden al classpath todas las librerias externas que haya en /var/www/
    shell_exec('mkdir out');
    $salida = secure_exec('javac -d '.$path.'out -Xlint:unchecked -encoding UTF-8 -cp .:/var/www/* @sources_list.txt 2>&1');
    error_log("Junit: ".$salida."\n", 3, "/var/www/moodledata/temp/filestorage/evalcode.log");
    if(strpos($salida,'error') || strpos($salida,'errors')){
        $returndata->grade = 0;
        $returndata->feedbackcomment = "Error during compilation (javac): <br>".str_replace("\n", "<br>", $salida);
        return $returndata;
    }

    $junitResult = secure_exec('java -jar /var/www/junit-platform-console-standalone-1.6.0.jar --class-path out --scan-class-path 2>&1');

    $output = $junitResult;
    $fh = fopen('feedback.log','w');
    fwrite($fh,$output);
    fclose($fh);

    $feedback= shell_exec('ansi2html --white < feedback.log');

    $countTotalTest="";
    $countSuccesfullTest="";
    $countFailureTest="";

    preg_match_all("/([0-9]+) tests found/",$junitResult,$countTotalTest);
    $countTotalTest = strtok($countTotalTest[0][0], " ");

    preg_match_all("/([0-9]+) tests successful/",$junitResult,$countSuccesfullTest);
    $countSuccesfullTest = strtok($countSuccesfullTest[0][0], " ");

    preg_match_all("/([0-9]+) tests failed/",$junitResult,$countFailureTest);
    $countFailureTest = strtok($countFailureTest[0][0], " ");
```

Figura 17: Fragmento de código de fichero "*evaluate.php*" de JUnit

La versión 5 de JUnit tiene además la ventaja de que es compatible con todas las versiones anteriores de JUnit, por lo que, tras esta actualización, la herramienta sigue siendo compatible con JUnit4 o con cualquier versión anterior que se desee utilizar.

3.2.3 Cambio en sistema de feedback

El desarrollo de la herramienta para corrección de estilo de código en lenguaje C, de la que hablaremos con detalle en el apartado 3.3.2, provocó un cambio en cuanto al sistema de *feedback* de EvalCode. Este cambió se llevó a cabo debido a dos problemas:

- Los *renderers* de Moodle provocan que cualquier salida en formato HTML, como la que se genera cuando el *plugin* EvalCode procesa la herramienta de estilo, no se pueda ver en su formato original en la casilla de comentario.
- Además de no respetar el formato, si se usan otras herramientas junto a esta, solo se muestra la salida en HTML, por lo que solo se muestra el *feedback* de la herramienta de estilo y se eliminan los demás.

Por estas razones y especialmente para permitir que cualquier futura herramienta pueda tener su salida en formato HTML, se llevó a cabo un cambio del sistema de *feedback*. El nuevo sistema de *feedback* propuesto genera una tabla en el apartado de comentarios. Esta tabla contiene un desglose de la calificación con los porcentajes de valor de cada herramienta y la nota que se ha obtenido en cada una de ellas. Por otro lado, el *feedback* generado por cada una de las herramientas ejecutadas se integrará en un único fichero de *feedback* que los alumnos podrán descargar en su equipo. En la figura 18 podemos apreciar el nuevo sistema con la tabla con el desglose de la calificación y el fichero de *feedback*.

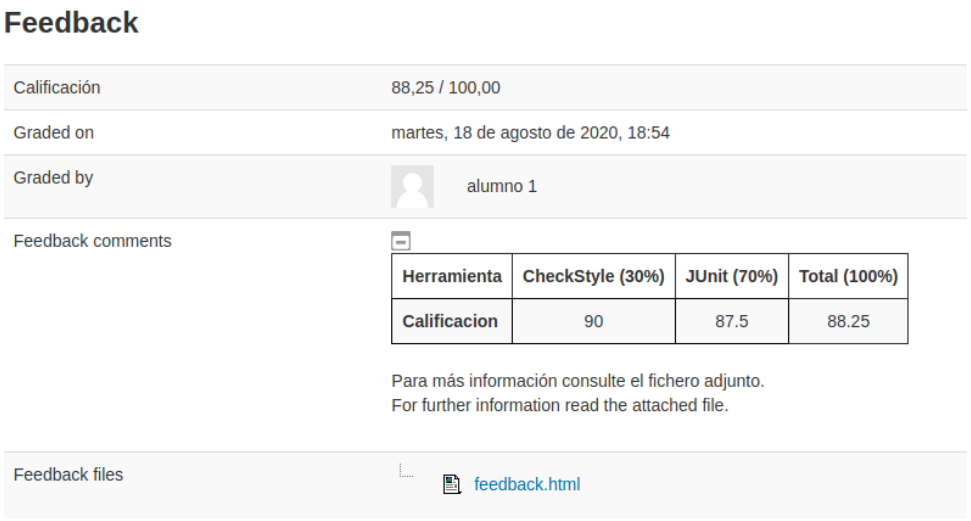


Figura 18: Ejemplo de *feedback* con comentario en formato tabla y fichero

Este nuevo sistema permitirá que sea más sencillo integrar nuevas herramientas en un futuro, ya que simplemente ejecutando cualquier *script* que proporcione una salida en HTML ya se podrá retornar la salida sin tener que procesar el texto. En caso de utilizar múltiples herramientas, la extensión EvalCode se encarga de concatenar todas las salidas e integrarlas en el fichero de *feedback*. Otro aspecto a tener en cuenta es que el formato HTML permite al profesor hacer un *feedback* más completo y vistoso para el alumno (por ejemplo, la creación de tablas o el uso de colores).

Aunque este sistema acepte cualquier salida, si se desactivan los ficheros de *feedback* desde el formulario de creación de una tarea EvalCode, el *feedback* se escribirá directamente en el comentario. Se han conservado las dos alternativas para que el *plugin* no falle si se desactiva esta opción.

Para este rediseño del sistema de evaluación se ha trabajado sobre la función "*process_save_submission*()", donde ahora, si está activada la opción de ficheros de *feedback*, creamos el fichero de *feedback* con la función "*create_file_from_string*()", de la File API y se devuelve como *feedback file*. También se crea, para el espacio de comentario, una tabla HTML que nos detallará el desglose de las notas por

herramientas. En la figura 19 podemos ver un pseudocódigo que detalla los procesos que se llevan a cabo en la función señalando en distinto color el código nuevo que se han introducido en la función.

```
función process_save_submission:
    inicia el proceso de evaluación de la entrega
    descargar XML con las herramientas de evaluación de la BD
        para cada herramienta de evaluación seleccionada hacer
            si la herramienta no tiene un script de evaluación
                lanza mensaje de error
            fin si
            guardar entrega en la BD
            crear un directorio temporal
            descargar y descomprimir entrega
            descargar y descomprimir ficheros de evaluación del profesor
            si la entrega tiene el formato correcto
                llama a función de evaluación de la herramienta: retorna nota y feedback
                multiplica la nota por el porcentaje asignado a esa herramienta
                guarda la nota y el feedback en una variable local
             si no
                lanza mensaje de error
             fin si
        fin para
    guarda la nota total y el feedback de todas las herramientas en la BD
    comprobar si están activados los ficheros de feedback
    si están activados los ficheros de feedback
        crear fichero con los feedbacks concatenados
        adjuntarlo como fichero de feedback
        crear tabla con los resultados de cada herramienta y retornarlo como comentario
    si no
        retornar feedback en la casilla de comentario
    fin si
    retorna la nota total de la tarea
    finaliza el proceso de evaluación
fin función
```

Figura 19: Pseudocódigo de la función "*process_save_submission*()" del *script* "*locallib.php*"

El fichero de *feedback* es un archivo de texto descargable que contendrá las salidas de los ficheros concatenadas separadas por un identificador de cada herramienta. Estas salidas pueden ser texto o HTML, siendo el *plugin* EvalCode responsable de integrarlas en el fichero definitivo. Esto se puede apreciar en el ejemplo de la figura 20 donde se ve el *feedback* de las herramientas Checkstyle y JUnit sin limitaciones de formato.

-CheckStyle (30%) feedback comment:

Errors: 0
Warnings: 1
Max errors permited: 5

Quality check grade: 80

Errors/Warns detected:
Comenzando auditoría...
[WARN] /var/www/moodledata/temp/filestorage/3_1598732956/pract03/ListaDobleEnlaceNoIter.java:3: Falta el comentario Javadoc. [JavadocType]
Auditoría concluida.

-JUnit (70%) feedback comment:

Total test(s): 11
Min correct test(s) required: 9
Failure test(s): 1

JUnit test grade: 75

```
Thanks for using JUnit! Support its development at https://junit.org/sponsoring

├─ JUnit Jupiter ✔
└─ JUnit Vintage ✔
   └─ ListaSimpleTest ✔
      ├─ testElimina ✔
      ├─ testAnhadePrimerElemento ✔
      ├─ testEliminaLimiteCapacidad ✔
      ├─ testBusca ✔
      ├─ testSuperaCapacidad ✔
      ├─ testHaceVacia ✘ Error: el tamanho deberia ser 0, pero es:1
      ├─ testAnhadePrimeraPos ✔
      ├─ testPosIncorrecta ✔
      ├─ testAnhade ✔
      ├─ testAnhadePosIntermedia ✔
      └─ testConstructor ✔
```

Figura 20: Ejemplo de *feedback* en fichero HTML

3.3. Nuevas funcionalidades

Una de las características principales de EvalCode es que permite a los profesores extender su funcionalidad y adaptarse al contexto de cada asignatura. En este apartado se describen las nuevas funcionalidades incorporadas en este proyecto.

3.3.1 Automatización de la matriculación de alumnos en el servidor

Las operaciones de mantenimiento y actualización sobre las herramientas ya implementadas, además de por criterios de calidad, estaban enfocadas a que el software estuviese preparado para ser probado de nuevo en la asignatura Métodos de Programación del Grado en Ingeniería Informática de la Universidad de Cantabria. Para que los alumnos dispongan de un usuario con el que hacer sus entregas de tareas en el servidor de pruebas en el que se desarrolla EvalCode, es necesario matricularlos a todos con un usuario y contraseña e incluirlos en el curso y en su correspondiente grupo de prácticas.

Para matricular a todos los alumnos de manera simultánea sin tener que ir de uno en uno Moodle da la opción de adjuntar un fichero de tipo “.csv” con un formato establecido por la plataforma. Estos ficheros “.csv” tienen la ventaja de que son editables mediante programas de hoja de cálculo, pudiendo hacer fácilmente columnas con los campos en forma de tabla. Los campos obligatorios para identificar a un usuario son *username* (nombre de usuario), *firstname* (nombre), *lastname* (apellido), *email* y *password* (contraseña). Además de estos, existen campos opcionales, que aportan más información del usuario, y los campos adicionales para la inscripción del usuario en

Moodle, de los que se van a usar el de curso, para matricularles en el curso Métodos de Programación 2020 y el de grupo, para dividir a los alumnos por grupos de prácticas.

Para conseguir la información de los alumnos de la Universidad de Cantabria es importante destacar que el campus virtual de esta institución dispone de una opción para generar un fichero de texto que contiene la lista de alumnos matriculados en cada asignatura. Mediante este fichero se puede generar el archivo ".csv" agrupando por campos los datos de los alumnos en una hoja de cálculo.

Para automatizar este proceso se escribió un *script* que procesa y convierte el listado de alumnos generado automáticamente a través del campus virtual a ".csv", rellenando todos los campos salvo el del grupo, que debe completarse manualmente. En la figura 21 podemos ver la función principal de este *script*, que prepara el fichero de texto para ser procesado, define las columnas del ".csv" y va extrayendo los datos (hay algunos que se generan con otros datos, como el usuario, que se extrae del *email*, y la contraseña, que es el número de DNI) para escribirlos en el fichero ".csv" en formato de hoja de cálculo.

```
#! /bin/bash
# Script_name     : evalcode_generate_moodle_csv_from_student_list.sh
# Author          : Evalcode
# Description     : Create a csv file to facilitate importing student accounts in moodle
# Version         : v1.0
# Prerequisites   : student list from campus, bash >= 4.2

function generate_moodle_list {
    TMP_FILE=list.csv
    SUFFIX="@alumnos.unican.es"
    SEPARATOR="," # should we use ; as $FULLNAME includes a comma?

    # Adapt source file
    dos2unix -q "${SOURCE_FILE}" > /dev/null # Files from campus usually have Windows codification for end of line
    sed -i '/^$/d' "${SOURCE_FILE}"          # Remove blank lines
    tail -n +2 "${SOURCE_FILE}" > $TMP_FILE  # Consider first line as header and omit it

    # Write csv header for moodle
    echo "username${SEPARATOR}password${SEPARATOR}firstname${SEPARATOR}lastname${SEPARATOR}email${SEPARATOR}course1${SEPARATOR}group1">$DEST_FILE

    echo "  Reading list and generating $DEST_FILE..."
    while IFS=";" read -r dni fullname email remainder # use 'remainder' to avoid ending inconsistencies (; or ;+blank)
    do
        email=$(echo "$email" | tr -d '\r') # must remove \r special character for new line
        dni=$(echo "$dni" | tr -d '\r')
        fullname=$(echo "$fullname" | tr -d '\r')
        name=${fullname##*,}             # will drop begin of string up to last occur of `,`
        name=$(echo "$name" | xargs)     # trim leading spaces
        surname=${fullname%,*}           # will drop part of string from last occur of `,` to the end
        id=$(basename -s $SUFFIX $email) # get id for moodle removing suffix from email
        echo "$id${SEPARATOR}$dni${SEPARATOR}$name${SEPARATOR}$surname${SEPARATOR}$email${SEPARATOR}$COURSE_NAME${SEPARATOR}">>$DEST_FILE
    done < $TMP_FILE
    echo "  Done"
    rm -f $TMP_FILE
}
```

Figura 21: Función *generate_moodle_list* del *shell script* de conversión a ".csv"

Este *script* se ejecuta desde línea de comandos y recibe como parámetros de entrada la dirección del fichero con el registro de alumnos y el identificador del curso. En la figura 22 tenemos un ejemplo de su uso. El fichero ".csv" se puede subir a Moodle en la pestaña de usuarios del menú de administrador, en la opción añadir nuevos usuarios.

```
./evalcode_generate_moodle_csv_from_student_list.sh Listado_de_Alumnos_de_la_Asignatura.txt MetProg
```

Figura 22: Ejemplo de comando de ejecución del *script*

3.3.2 Herramienta de revisión de estilo para lenguaje C

Otra nueva funcionalidad que se incorporará al *plugin* EvalCode en este proyecto será una herramienta de corrección de estilo para lenguaje C. La herramienta elegida para realizar esta tarea es Astyle [18], un software de corrección de estilo compatible con varios lenguajes de programación. Para facilitar su integración con EvalCode, la herramienta Astyle se utilizará a través de una segunda herramienta denominada Style50 [19], un programa específico para lenguaje C que utiliza Astyle para hacer las correcciones de estilo y proporciona una salida más visual en formato HTML, marcando el código con colores señalando en rojo las partes a eliminar y en verde las que se deben añadir. Asimismo, esta herramienta tiene la particularidad de que genera su propia calificación.

La herramienta aprovecha la mejora que permite realizar entregas de ficheros sin comprimir, pudiendo procesar la herramienta entregas de archivos ".c". El software solo admite un fichero ".c" por entrega.

Una particularidad de esta herramienta es que no es integrable directamente en Moodle de la misma forma que las otras herramientas integradas en el *plugin*, por lo que para ella hemos usado el nuevo sistema de *feedback* implementado en EvalCode, del que se ha hablado en el apartado 3.2.3. En la figura 23 se puede ver un ejemplo del *feedback* en formato HTML que genera la herramienta.

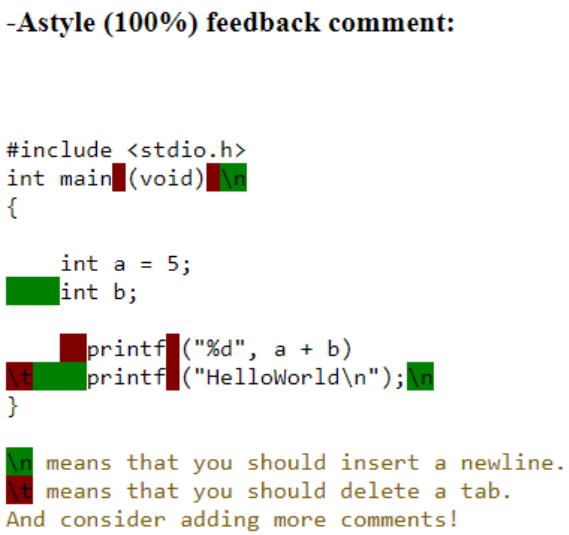


Figura 23: Ejemplo de *feedback* en fichero de Style50

Esta herramienta ya estaba planteada desde la versión "1.0", aunque únicamente se llegó a crear el directorio y los *scripts* en la carpeta "*tools*/", pero en forma de prototipo, por lo que estos carecían de funcionalidad. El trabajo consistió en estudiar el uso y ejecución de la herramienta Style50 e implementar su integración en el fichero "*evaluate.php*", uno de los ficheros característicos de cada herramienta (ver apartado 3.1.3).

En el fichero "*evaluate.php*" es donde se realiza la ejecución y procesado de la herramienta. En la figura 24 podemos ver el código de la función $*evaluatefunc* que es la que realiza las operaciones. Se utilizan dos ejecuciones de comandos en la *shell*, una para obtener la nota que otorga la herramienta y otra para obtener el *feedback*. Dado que la salida de esta herramienta es textual, pero incluyendo códigos ANSI para el formato, es necesario procesarla para generar un formato compatible con EvalCode. En este caso, para mantener el formato de la salida original en el *feedback* generado, se optó por procesar la salida de la herramienta de estilo y convertirla a HTML. Para obtener la calificación, se multiplica por cien la nota que otorga Style50 para que se adecue al formato de notas que otorgan las demás herramientas. A diferencia de las otras herramientas integradas, la herramienta de estilo no utiliza los parámetros adicionales de evaluación para el cálculo de la nota, ya que la herramienta genera su propia calificación.

```
$evaluatefunc = function ($path,$returndata,$additionalParams){
    global $USER;

    secure_exec('find * -name "*.c" > sources_list.txt');
    $contents = file_get_contents('sources_list.txt');

    //It only takes the first file. Only working for 1 file
    $salida = shell_exec('style50 -o score '.$contents);

    if(strpos($salida,'error') || strpos($salida,'errors')){
        $returndata->grade = 0;
        $returndata->feedbackcomment = "Error: <br>".str_replace("\n", "<br>", $salida);
        return $returndata;
    }

    $grade = floatval($salida)*100;
    $output = secure_exec('style50 '.$contents);
    $fh = fopen('feedback.log','w');
    fwrite($fh,$output);
    fclose($fh);

    $feedback= shell_exec('ansi2html --white --title "'.fullname($USER).'" < feedback.log');

    $fh = fopen('feedback.html','w');
    fwrite($fh,$feedback);

    $comment = "".$feedback;

    $returndata->grade = $grade;
    $returndata->feedbackcomment = $comment;
    return $returndata;
};
```

Figura 24: Función $*evaluatefunc* del *script* "*evaluate.php*" de la herramienta Astyle

3.3.3 Herramienta de ejecución segura de código externo

Un aspecto crucial en cualquier software es la seguridad y más en un proyecto como EvalCode, que ejecuta código desarrollado por personas externas al proyecto en el servidor de explotación. Por esta razón se decidió implementar una herramienta de tipo *sandbox* que asegurase una ejecución segura del código de los alumnos. Un *sandbox* se refiere a un entorno controlado en el que aislar la ejecución de un programa para evitar

que cualquier *malware* o software malicioso pueda tener acceso a ningún fichero del sistema salvo los estrictamente necesarios para la ejecución.

El programa de tipo *sandbox* elegido para realizar esta función en EvalCode es el software Firejail [20], que reduce el riesgo de brechas de seguridad restringiendo el entorno de ejecución de aplicaciones usando los espacios de nombres de Linux. Este software, escrito en lenguaje C, se ejecuta en cualquier sistema Linux con una versión de *kernel* igual o mayor a la 3.x. Tras investigar y evaluar distintas alternativas, como SELinux [30], finalmente se optó por la herramienta Firejail por ser ligera, fácil de configurar y disponer de bastante información y ejemplos de uso. Una de las ventajas que tiene esta aplicación es que es fácilmente configurable mediante perfiles de seguridad. Estos perfiles son diferentes configuraciones que dictan qué partes del sistema o qué funciones están restringidas durante la ejecución de una aplicación y cuáles están disponibles, y con qué permisos (lectura, lectura-escritura…).

Hay diferentes opciones para los perfiles, como hacer *blacklist*, que protege un fichero o un directorio no permitiendo acceder a él mientras se ejecuta el *sandbox*. También se puede hacer *whitelist* (dar acceso a un fichero o directorio en concreto) o fijar ficheros o directorios como *read-only* o *read-write*, además de otros parámetros.

La herramienta cuenta con perfiles predeterminados ya diseñados para aplicaciones comunes o algunos genéricos que se pueden incluir en otros perfiles. En el caso de EvalCode, se va a crear un perfil nuevo basado en aquellos perfiles predeterminados que deshabilitan la mayoría de funciones y directorios del sistema para que la ejecución sea segura. Cada uno de estos perfiles se encarga de deshabilitar ciertas cosas:

- El perfil "**disable-common.inc**" deshabilita la mayoría de archivos y directorios del sistema. Un ejemplo de lo que limita este perfil es que fija como *read-only* los ficheros necesarios para ejecutar comandos que necesitamos para las ejecuciones, pero que igualmente queremos proteger. Otro ejemplo sería que pone en *blacklist* todos los ficheros relacionados con el arranque del sistema.
- El "**disable-devel.inc**" deshabilita las herramientas de desarrollo como Python, PHP u otras.
- El perfil "**disable-programs.inc**" contiene una lista muy extensa de programas a los que se haría *blacklist* en caso de que estuvieran instalados.
- Y, por último, el "**disable-passwdmgr.inc**" deshabilita software dedicado a la gestión de *passwords*.

En este proyecto se ha creado un perfil propio adecuado a las necesidades de EvalCode que hemos llamado "*evalcode.profile*". Este perfil se configura para que nos permita escribir en el directorio de operaciones "*/var/www/moodledata/temp/filestorage*". Este directorio situado dentro de la carpeta "*moodledata*/", de la que hablamos en el apartado 2.2, necesita permisos de escritura ya que es el que contiene los directorios temporales en los que se descargan todos los ficheros de una ejecución de EvalCode y donde se

realizan operaciones con ellos. También se configura Firejail para que no escriba su salida propia por consola cuando se invoca la ejecución del proceso, ya que no debe contaminar el *feedback* de las herramientas. Otras características a tener en cuenta para configurar el Firejail son que no permite ejecutar un comando utilizando "*sudo*" para pedir permisos de superusuario, o que para la depuración de fallos puede ofrecer información extra utilizando el parámetro "*--debug*".

La integración de esta herramienta en Moodle modifica el modo de ejecución de las herramientas de evaluación. Anteriormente, en los *scripts* "*evaluate.php*", se utilizaba únicamente la función PHP "*shell_exec*('comando')" con el comando de ejecución de la herramienta. Para usar el *sandbox* en la ejecución del código externo en EvalCode se ha creado un nuevo *script* denominado "*secure_exec.php*". Este nuevo *script*, que se encuentra en la carpeta "*tools/*" de EvalCode, contiene la función "*secure_exec*($comando)", con la que se utilizará el entorno de ejecución seguro. Está función realiza la ejecución de la herramienta mediante Firejail, como se puede ver en la figura 25.

```
function secure_exec(String $command) {
    return shell_exec('firejail --writable-var
      --profile=/var/www/html/moodle/mod/evalcode/tools/evalcode.profile --quiet '.$command);
}
```

Figura 25: Función "*secure_exec*()"

Para poder usar esta función "*secure_exec*()" es necesario importar el fichero "*secure_exec.php*" en los *scripts* PHP de ejecución, añadiendo la línea de código que se ve en la figura 26. A partir de este momento, siempre que se opere con ficheros de entregas de alumnos, se deberá llamar a esta función.

```
require_once('/var/www/html/moodle/mod/evalcode/tools/secure_exec.php');
```

Figura 26: Importación del *script* "*secure_exec.php*"

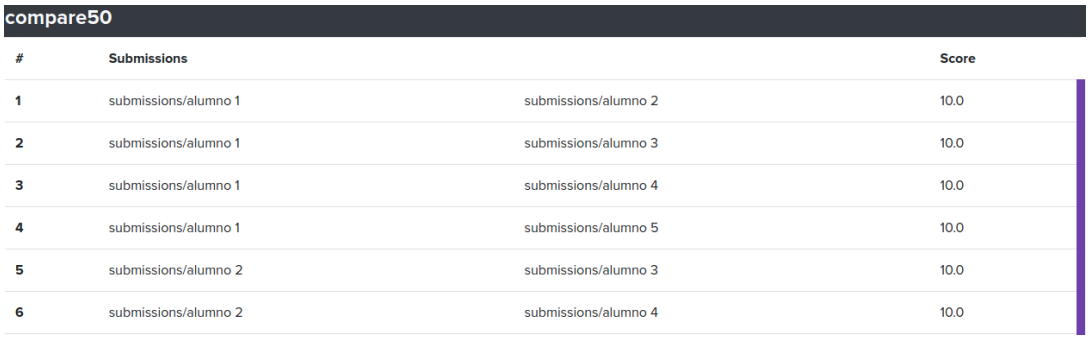
Aunque todos los perfiles de Firejail se encuentren de manera predeterminada en un directorio llamado "*profiles/*" dentro de la carpeta de instalación de la herramienta *sandbox*, en el Firejail todo se usa con rutas absolutas, así que a efectos prácticos para la ejecución es igual tener el perfil en la carpeta "*profiles/*" o dentro del módulo EvalCode. Por esta razón el perfil "*evalcode.profile*" se encuentra en la carpeta "*tools/*", facilitando así la instalación y exportación del *plugin*.

3.3.4 Herramienta de control de plagios

Otra funcionalidad nueva es la integración de una herramienta de detección de plagios en ejercicios de programación. La herramienta elegida para este cometido es

Compare50 [21], una herramienta que puede ser ejecutada por línea de comandos y que además proporciona una salida en formato HTML.

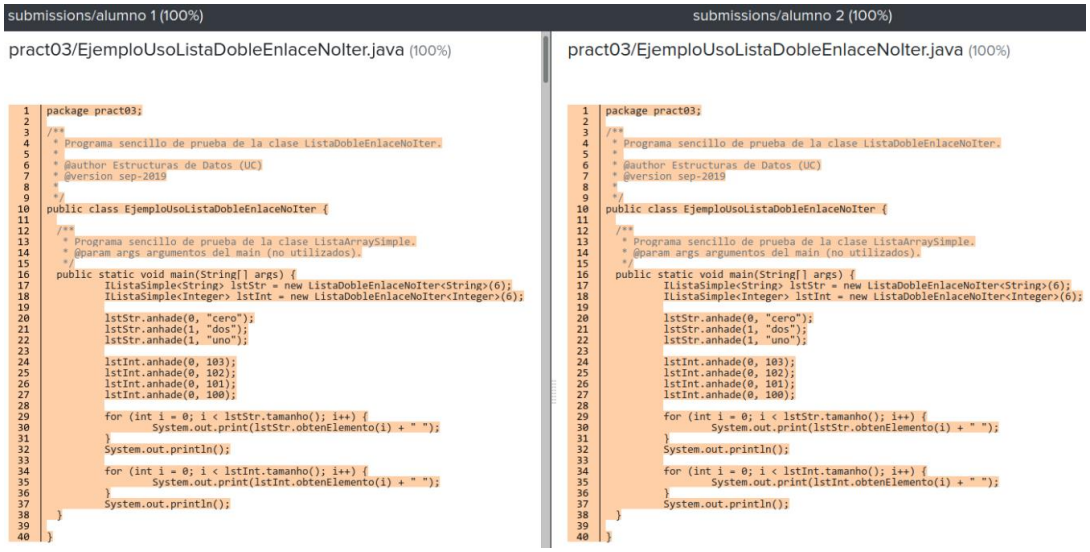
La herramienta genera un archivo "*index.html*", que funciona a modo de tabla en la que se ven las entregas con más similitudes entre sí, y un fichero por cada par de entregas comparadas, llamadas "*match_1.html*", "*match_2.html*" … sucesivamente. Para facilitar la lectura de los resultados, la salida de la herramienta de control de plagios se trunca a las 15 parejas que tienen más probabilidad de ser plagio. En las figuras podemos ver un ejemplo de un resultado de Compare50, viendo en la figura 27 el "*index.html*" con la tabla de comparaciones entre entregas ordenadas por el grado de similitud. Si pulsamos en una de esas comparaciones nos mostrará una tabla de dos columnas con los archivos de los alumnos y las coincidencias entre esos ficheros, resaltando las coincidencias en el texto, como se puede ver en la figura 28, siendo en ese caso dos archivos exactamente iguales.



compare50

#	Submissions		Score
1	submissions/alumno 1	submissions/alumno 2	10.0
2	submissions/alumno 1	submissions/alumno 3	10.0
3	submissions/alumno 1	submissions/alumno 4	10.0
4	submissions/alumno 1	submissions/alumno 5	10.0
5	submissions/alumno 2	submissions/alumno 3	10.0
6	submissions/alumno 2	submissions/alumno 4	10.0

Figura 27: Vista de un ejemplo de fichero "*index.html*" generado por la herramienta Compare50



submissions/alumno 1 (100%) — submissions/alumno 2 (100%)

pract03/EjemploUsoListaDobleEnlaceNoIter.java (100%)

```
package pract03;

/**
 * Programa sencillo de prueba de la clase ListaDobleEnlaceNoIter.
 *
 * @author Estructuras de Datos (UC)
 * @version sep-2019
 *
 */
public class EjemploUsoListaDobleEnlaceNoIter {

  /**
   * Programa sencillo de prueba de la clase ListaArraySimple.
   * @param args argumentos del main (no utilizados).
   */
  public static void main(String[] args) {
        IListaSimple<String> lstStr = new ListaDobleEnlaceNoIter<String>(6);
        IListaSimple<Integer> lstInt = new ListaDobleEnlaceNoIter<Integer>(6);

        lstStr.anhade(0, "cero");
        lstStr.anhade(1, "dos");
        lstStr.anhade(1, "uno");

        lstInt.anhade(0, 103);
        lstInt.anhade(0, 102);
        lstInt.anhade(0, 101);
        lstInt.anhade(0, 100);

        for (int i = 0; i < lstStr.tamanho(); i++) {
              System.out.print(lstStr.obtenElemento(i) + " ");
        }
        System.out.println();

        for (int i = 0; i < lstInt.tamanho(); i++) {
              System.out.print(lstInt.obtenElemento(i) + " ");
        }
        System.out.println();
  }
}
```

Figura 28: Vista de un ejemplo de fichero "*match_1.html*" generado por la herramienta Compare50

El mayor problema de la integración de esta herramienta es que no nos ha permitido reutilizar la estructura modular de integración de herramientas por el hecho de ser una herramienta diseñada exclusivamente para ser usada por el profesor, no para dar una retroalimentación inmediata a los alumnos. En este caso, el profesor ejecuta explícitamente esta herramienta a través de la interfaz gráfica y recibe los resultados con un sistema diferente al utilizado por los alumnos, ya que no nos sirven los mecanismos de *feedback* que poseen las entregas.

Por ello, se ha modificado EvalCode para que esta herramienta pueda ejecutarse desde la vista que tiene el profesor de todas las entregas de los alumnos, de la que ya hemos hablado en el apartado 3.1.3. Para que el profesor inicie la ejecución de la herramienta, se decidió incorporar una nueva opción en un formulario de menú desplegable que podemos ver en la figura 29.

Figura 29: Vista del formulario *select* con el que se ejecuta la herramienta Compare50

Este formulario está definido en el fichero "*gradingbatchoperationsform.php*". En la figura 30 se puede ver el método "*definition*()" de este formulario, en el que se ha añadido una nueva opción para el elemento de formulario de tipo *select*.

```php
public function definition() {
    $mform = $this->_form;
    $instance = $this->_customdata;

    // Visible elements.
    $options = array();

    $options['runevalcode'] = get_string('runevalcode', 'evalcode');
    $options['runcompare'] = get_string('runcompare', 'evalcode');

    $mform->addElement('hidden', 'action', 'gradingbatchoperation');
    $mform->setType('action', PARAM_ALPHA);
    $mform->addElement('hidden', 'id', $instance['cm']);
    $mform->setType('id', PARAM_INT);
    $mform->addElement('hidden', 'selectedusers', '', array('class'=>'selectedusers'));
    $mform->setType('selectedusers', PARAM_SEQUENCE);
    $mform->addElement('hidden', 'returnaction', 'grading');
    $mform->setType('returnaction', PARAM_ALPHA);

    $objs = array();
    $objs[] =& $mform->createElement('select', 'operation', get_string('chooseoperation', 'evalcode'), $options);
    $objs[] =& $mform->createElement('submit', 'submit', get_string('go'));
    $batchdescription = get_string('batchoperationsdescription', 'evalcode');
    $mform->addElement('group', 'actionsgrp', $batchdescription, $objs, ' ', false);
}
```

Figura 30: Método "*definition*()" del formulario "*gradingbatchoperationsform.php*"

En lo que respecta al resultado, se planteó inicialmente que el profesor pudiera acceder directamente desde el navegador a los ficheros HTML que genera la herramienta. Sin embargo, esta opción no fue posible ya que el directorio "*moodledata*/", dónde se encuentran esos ficheros, está fuera de los directorios accesibles por el servidor Apache, por lo que se optó por generar un ".zip" y retornarlo como fichero al profesor, tal y como se ilustra en la figura 31.

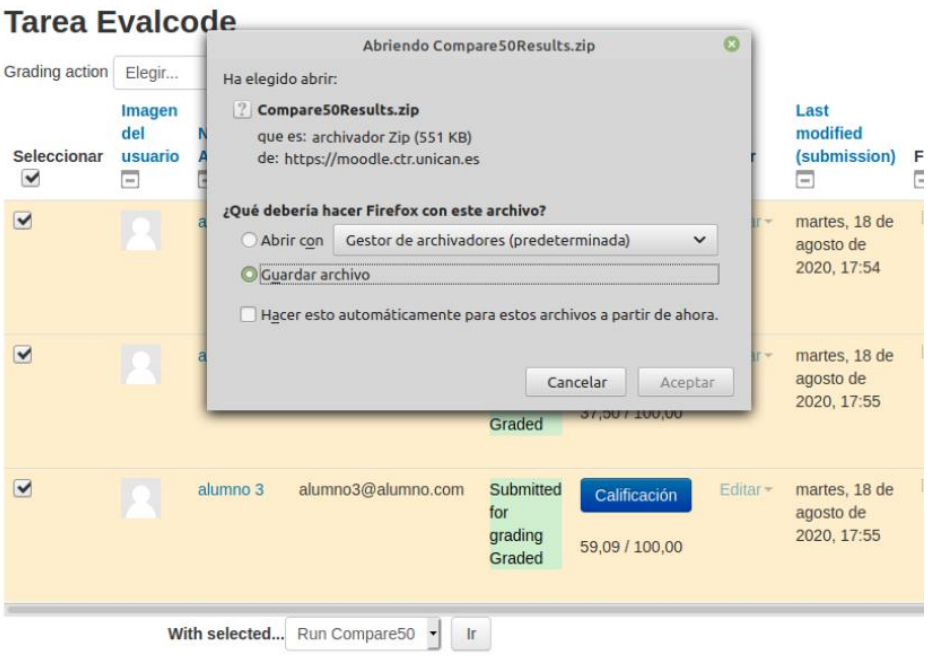


Figura 31: Figura de ejemplo de la descarga que se genera al ejecutar Compare50

Otra característica importante de la herramienta de control de plagios es la posibilidad de incorporar un mecanismo para poder añadir un conjunto de archivos que no se tienen en cuenta en la revisión de plagio. Por ejemplo, esto resulta útil cuando un profesor entrega partes de un programa ya implementado a los alumnos. Esta característica es soportada por la herramienta Compare50, pero necesita ser integrada en EvalCode.

Para desarrollar esta característica la cuestión era cómo y dónde adjuntar estos ficheros de profesor. Tras evaluar varias alternativas se decide incorporar un nuevo elemento *filemanager* en el formulario de creación de actividad "*mod_form.php*", tal y como se puede ver en la figura 32. Este elemento *filemanager* irá a continuación de los otros dos ya existentes en el formulario. En la figura 33 podemos ver la vista de los tres elementos *filemanager* con ficheros adjuntos.

```
$mform->addElement('filemanager', 'introattachments',
    'Provided Files',
    null, array('subdirs' => 0, 'maxbytes' => $COURSE->maxbytes));

$mform->addElement('filemanager', 'introattachmentsjunit',
    'Evaluation Files',
    null, array('subdirs' => 0, 'maxbytes' => $COURSE->maxbytes));

//Plagiarism teacher files
$mform->addElement('filemanager', 'plagiarismteacherfiles',
    'Files for plagiarism tool',
    null, array('subdirs' => 0, 'maxbytes' => $COURSE->maxbytes));
```

Figura 32: Fragmento del formulario "*mod_form.php*" correspondiente a la definición de los elementos *filemanager* que se usarán en la creación de actividad

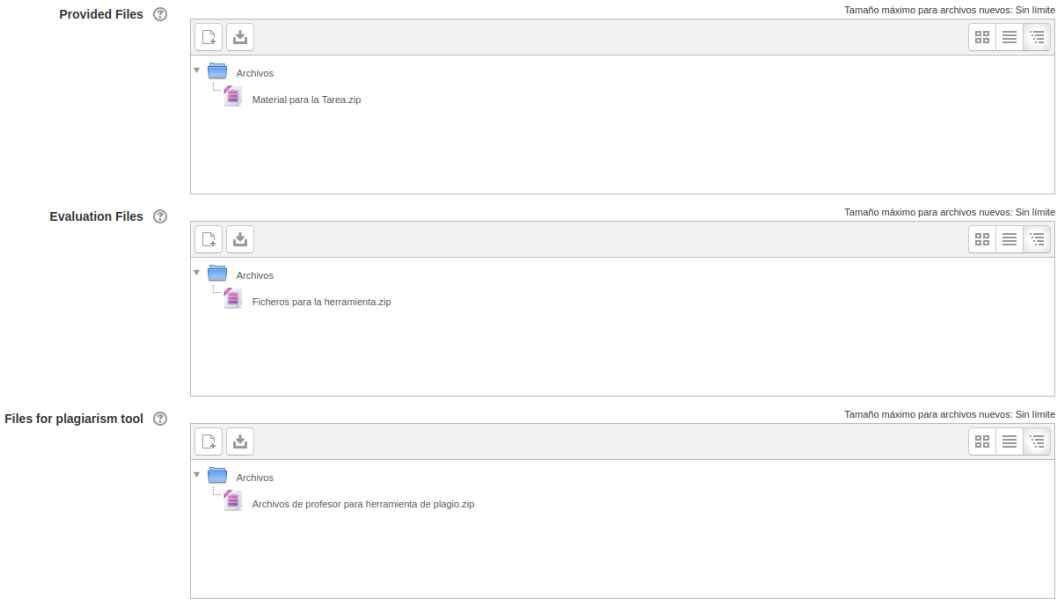


Figura 33: Vista de los tres elementos *filemanager* del "*mod_form.php*"

El código de ejecución de esta nueva herramienta no puede estar en la carpeta "*tools/*", ya que no funciona de la misma manera que las demás herramientas y no puede ser detectada como una más. Por ello se ha ubicado el código correspondiente a la ejecución en una nueva carpeta dentro de EvalCode llamada "*plagiarism/compare50/*". El código de llamada a la ejecución de la herramienta de control de plagio se encuentra en la función "*process_grading_batch_operation()*", de la que ya hemos hablado en el apartado 3.1.3. En la figura 34 se puede ver el pseudocódigo de la parte del código del método que se ha añadido.

```
función process_grading_batch_operation:
    …
    si operación elegida es igual a Run Compare50
        crear carpeta temporal
        si hay archivos del profesor
            crear directorio files/
            descargar archivos de profesor a directorio files/
        fin si
        crear directorio submissions/
        para cada alumno seleccionado
            crear directorio con nombre del alumno
            si el alumno ha realizado la entrega
                descargar entrega en directorio del alumno
            fin si
        fin para
        llamada a función de ejecución de Compare50
    fin si
fin función
```

Figura 34: Pseudocódigo de parte de la función "*process_grading_batch_operation*()"

Los ficheros de los alumnos se descargan en un directorio específico para cada ejecución dentro de "*moodledata*/". El directorio contendrá a su vez 3 carpetas: una llamada "*submissions*/", que contendrá las entregas de los alumnos divididas por directorios con el nombre del alumno; otra llamada "*files*/", donde se descargarán los ficheros del profesor, si se ha adjuntado alguno; y un directorio "*results*/", con los resultados de la ejecución. Para dar el resultado se comprime el directorio "*results*/" en un ".zip" y se genera la descarga de la figura 31. En la figura 35 podemos apreciar gráficamente la estructura de este directorio temporal.

Figura 35: Ejemplo de jerarquía del directorio de ejecución de Compare50

Uno de los problemas más relevantes en la integración de esta herramienta fue que el Compare50 fallaba inexplicablemente cuando se ejecutaba desde PHP con la función "*shell_exec*()", mientras que ejecutándola directamente en la *shell* no daba ningún problema. El motivo era que existía un problema en la ejecución debido a que Apache no utilizaba los archivos de configuración de idioma (*locales*) configurados por defecto en el sistema. Para solucionar este error y que Apache utilice por defecto los *locales* del sistema, se modificó el archivo de configuración de Apache, "*/etc/apache2/envvars*".

En la figura 36 podemos ver el código de la función "*executeCompare*()", que lleva a cabo la ejecución de Compare50 comprobando si se han adjuntado o no ficheros del profesor y ejecutando un comando distinto en cada caso. Estos comandos se ejecutan teniendo en cuenta la jerarquía de carpetas que se ha explicado anteriormente. Tras la ejecución, en la misma función se comprime la carpeta "*results*/" en un archivo ".zip" y se fuerza la descarga de ese fichero comprimido en el navegador. Esta descarga se

genera mediante la función "*header*()" de PHP, que permite enviar encabezados HTTP a un cliente web.

```php
//File that contains the secure_exec function to operate in a sandbox with the submmited files
require_once('/var/www/html/moodle/mod/evalcode/tools/secure_exec.php');

/**
 * This file contains the function executeCompare
 */

 /**
 * This is the function that is called to execute the plagiarism tool compare50 and return it
 * in a forced download
 * @param $path path to the students files folder
 *
 */

function executeCompare($path) {
    chdir($path);
    chdir('..');

    //If the teacher has given any files we execute the command with the -d flag
    if (file_exists('files/')){
        $command="compare50 ./submissions/* -d files/ -n 15";
    }else{
        $command="compare50 ./submissions/* -n 15";
    }

    $result = secure_exec($command);

    //We compress the results folder
    shell_exec('zip -r results.zip results');

    //We force a download to return the result
    $file = "results.zip"; //Name of the file to look for
    $filename = "Compare50Results.zip"; //Name of the file in the download
    $filepath = $path .'/../'. $file;

    // Process download
    if(file_exists($filepath)) {
        ob_start();

        header("Content-type: application/force-download");
        header('Content-Disposition: attachment; filename="'.$filename.'"');
        header('Content-Length: ' . filesize($filepath));

        while (ob_get_level()) {
            ob_end_clean();
        }

        readfile($filepath);
    }
}
```

Figura 36: Función "*executeCompare*()" en el fichero "*execute.php*"

4. Pruebas

En el proceso de desarrollo de cualquier software y más en uno complejo en el que no se tiene experiencia de desarrollo, como Moodle, es importante la realización de pruebas que aseguren el correcto funcionamiento del sistema.

Debido a la situación provocada por el COVID-19 no se pudieron realizar pruebas en el servidor de explotación de la herramienta hasta ya finalizado el desarrollo, ya que se dejó de tener acceso a él. Esto provocó que algunas de las pruebas no se completaran de la forma que se plantearon inicialmente.

4.1. Pruebas para las Herramientas

Un importante punto a la hora de desarrollar o realizar mantenimiento en cualquier herramienta es hacer las debidas pruebas para comprobar que se está obteniendo el comportamiento deseado. A lo largo del desarrollo realizamos pruebas en las siguientes herramientas:

- Herramientas para lenguaje Java

  Se realizaron pruebas cuando se llevaron a cabo actualizaciones en la herramienta JUnit, cuando se hizo el cambio en el sistema de *feedback* y cuando se prepararon tareas para probar el software con alumnos.

  Para diseñar estas pruebas se usaron tareas diseñadas en años anteriores para la asignatura Métodos de Programación del Grado de Informática. El código de estas tareas fue modificado mediante el entorno de desarrollo Eclipse para provocar fallos abarcando todos los posibles casos de uso de las herramientas JUnit y Checktyle.

- Herramienta de corrección de estilo en lenguaje C

  Para las pruebas de la herramienta de corrección de estilo en lenguaje C, se diseñaron unos sencillos programas en C para evaluar su comportamiento. Se crearon varios variando los fallos de estilo presentes. Esta herramienta, al generar su propio HTML e incluso su propia calificación, no necesitó de tantas pruebas sobre el funcionamiento de la herramienta en sí como del procesado del resultado.

  Haciendo pruebas cuando se desarrollaba esta herramienta es cuando se detectó el problema con el *feedback* en HTML que proporciona Style50 y a partir de aquí se decidió hacer todo el rediseño del sistema de *feedback* para pasar a devolver el resultado mediante un fichero (ver apartado 3.2.3). Para probar todo

este cambio de sistema se usó siempre esta herramienta, ya que era la que daba problemas con la salida en HTML.

4.1.1 Pruebas para herramienta de ejecución segura de código externo

Las pruebas realizadas para esta nueva funcionalidad estaban enfocadas a comprobar si realmente otorga las prestaciones de seguridad que se buscaban con su integración. Para ello, se diseñó una prueba en la que en una ejecución de un programa Java para la herramienta JUnit se ejecutase un comando de *shell* de lectura de un fichero protegido que el Firejail no debería permitir. El fichero elegido fue el "*/etc/passwd-*", un fichero que se utiliza en la administración de contraseñas en sistemas Unix. Este fichero está expresamente marcado como *blacklist* en el perfil "*disable-common.inc*", por lo que no se debería poder acceder a él mientras se ejecuta la herramienta. En la figura 37 se puede ver el fragmento de código que se ha incluido en una entrega de una tarea en lenguaje Java para realizar la prueba. En este código se ejecuta un comando de *shell* que lee este fichero y pintamos por pantalla tanto la salida normal como la salida de errores.

```
    try {
        //Comando de lectura de fichero de passwords
        String cmd = "cat /etc/passwd-";
        System.out.println("\n\nResultado ejecución comando 'cat /etc/passwd-':\n\n" + execCmd(cmd) + "\n");
    } catch (IOException ioe) {
        System.out.println (ioe);
    }
}

public static String execCmd(String cmd) throws java.io.IOException {
    java.util.Scanner a = new java.util.Scanner(Runtime.getRuntime().exec(cmd).getInputStream()).useDelimiter("\\A");
    java.util.Scanner s = new java.util.Scanner(Runtime.getRuntime().exec(cmd).getErrorStream()).useDelimiter("\\A");
    String valInput = "";
     if (a.hasNext()) {
        valInput = a.next();
     }
     else {
        valInput = "";
    }
     String valError = "";
     if (s.hasNext()) {
        valError = s.next();
     }
     else {
        valError = "";
    }
    return valInput + valError;
}
```

Figura 37: Fragmento de código Java utilizado para ejecutar un comando de *shell*

En la prueba se realizan dos ejecuciones, una con "*shell_exec*()", como se hacía anteriormente y otra llamando a la función "*secure_exec*()". El resultado, que se puede observar en la figura 38, es que en la primera deja leer el fichero sin problema mientras que en la segunda se recibe una notificación de que no se tienen permisos para leer ese fichero. Con esta prueba, en la que se ha obtenido el comportamiento deseado, comprobamos que el Firejail nos otorga las prestaciones de seguridad esperadas.

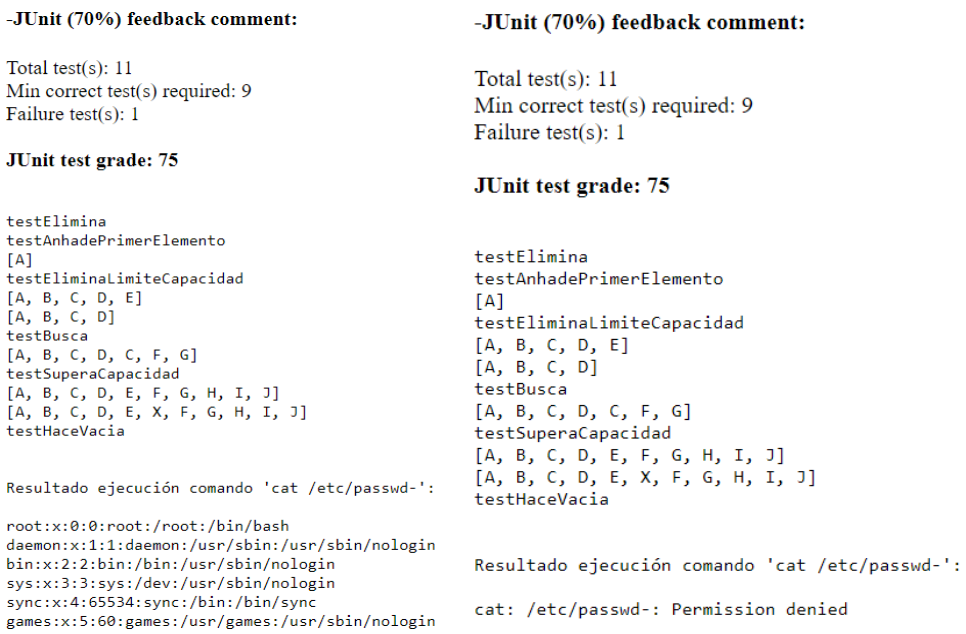


Figura 38: Ejemplo de *feedback* de la ejecución de la prueba con "*shell_exec*()" (izquierda) y con "*secure_exec*()" (derecha)

4.1.2 Pruebas para herramienta de detección de plagios

Para las pruebas de esta herramienta afectó considerablemente la situación del COVID-19, ya que en el servidor de explotación era donde se tenían numerosas entregas de alumnos de Métodos de Programación que se podían reutilizar para pruebas reales de detección de plagio. En las pruebas realizadas durante el desarrollo, se utilizaron hasta tres alumnos con entregas similares, para que la herramienta las comparase. De esta manera la herramienta funcionaba perfectamente, pero no se había probado con una gran cantidad de alumnos, ya que no se tenía acceso al servidor con entregas reales. Cuando se pudieron realizar pruebas en el servidor, se descubrió que había un fallo en las entregas con más de 10 alumnos, ya que Moodle divide la vista de las entregas por páginas cuando se superan esta cantidad de alumnos y la herramienta solo funciona con las entregas que se seleccionan para comparar en una misma vista.

El módulo *Tarea* original de Moodle, en la vista de todas las entregas, tiene en un panel de opciones que otorga la posibilidad de listar todos los alumnos en la misma página, siendo esta opción una solución sencilla al problema detectado. Sin embargo, el panel de opciones que otorga esta posibilidad no fue incluido en EvalCode. Esta decisión fue tomada en las primeras etapas de desarrollo (seguramente en la versión "0.1") sin prever que pudiera ser necesario en un futuro. Por cuestiones de tiempo, este apartado se dejará como trabajo futuro y será una de las tareas de mantenimiento más prioritarias a realizar para que la herramienta de detección de plagios sea plenamente funcional.

4.2. Pruebas en operación con alumnos reales

En cuanto a las pruebas en operación con alumnos reales, debido a la interrupción de las clases por la situación relativa al COVID-19, solo se pudieron realizar pruebas en dos prácticas de la asignatura Métodos de Programación del Grado en Ingeniería Informática de la Universidad de Cantabria.

Como ya comentamos en el apartado 3.2.2, de estas pruebas surgió la necesidad de hacer actualizaciones en la herramienta JUnit debido a que en anteriores cursos se usaba JUnit4 en vez de JUnit5.

En las dos prácticas en las que se pudieron realizar pruebas el software evaluaba satisfactoriamente las tareas, pero había claros problemas de rendimiento, especialmente cuando se conectaban varios alumnos a la vez. Esto es debido a que el servidor de explotación original se ejecutaba en un ordenador demasiado antiguo.

Por esta razón se decidió migrar el servidor a un ordenador más potente, con un procesador más actualizado y un disco de estado sólido o SSD. En la tabla 2, que contiene los tiempos de ejecución de algunas de las herramientas, se pueden apreciar las diferencias de rendimiento obtenidas en este cambio. Además, en la tabla 3 se puede ver que esta mejora de rendimiento es todavía más clara cuando se hacen varias entregas a la vez. En este caso el servidor nuevo incrementa su tiempo de ejecución, pero se mantiene dentro de unos márgenes que creemos aceptables, mientras los tiempos se incrementan fuertemente en el servidor original.

Herramienta	**Tiempo de ejecución en servidor antiguo (s)**	**Tiempo de ejecución en servidor nuevo (s)**
Astyle	2,914	1,151
JUnit y Checkstyle	8,058	3,386

Tabla 2: Tabla de tiempos de ejecución de herramientas

Herramienta	**Tiempo de ejecución en servidor antiguo (s)**	**Tiempo de ejecución en servidor nuevo (s)**
Astyle	Entrega1 - 6,678 Entrega2 - 7,656 Entrega3 - 7,201	Entrega1 - 1,015 Entrega2 - 1,163 Entrega3 - 1,550
JUnit y Checkstyle	Entrega1 - 26,152 Entrega2 - 24,520 Entrega3 - 27,064	Entrega1 - 8,417 Entrega2 - 8,418 Entrega3 - 7,659

Tabla 3: Tabla de tiempos de ejecución de herramientas con 3 entregas simultáneas

5. Conclusiones y trabajo futuro

En este proyecto se ha llevado el cabo el mantenimiento y la implementación de nuevas funcionalidades en un sistema de evaluación automática integrado en la plataforma de gestión de aprendizaje Moodle.

Este software de extensión de Moodle, llamado EvalCode, contenía en su versión preliminar dos herramientas de evaluación automática para lenguaje Java, JUnit y Checkstyle, que se han mejorado y actualizado para incrementar su usabilidad. Las mejoras generales más destacables para el sistema han sido: la actualización de la herramienta JUnit de su versión 4 a su versión 5; la refactorización del sistema de exposición de los resultados; la integración de una herramienta de corrección de estilo en lenguaje C basada en Astyle; la implementación de un método de ejecución segura mediante el software de *sandboxing* Firejail; y la implementación de una herramienta de detección de plagios, llamada Compare50. Todas estas novedades son plenamente funcionales salvo la herramienta de detección de plagios, que presenta actualmente un fallo que impide su uso en entornos con un número elevado de alumnos. Dicho fallo no ha sido corregido en este proyecto por falta de tiempo, aunque en esta memoria se propone una solución al respecto.

El software queda en un estado completamente funcional para poder ser utilizado en las asignaturas de programación de la Universidad de Cantabria.

Por último, se enumeran a continuación alguna de las líneas de trabajo futuro más relevantes dentro del proyecto EvalCode:

- Ampliar las restricciones de la herramienta *sandbox* aplicando al entorno seguro de ejecución limitaciones también en aspectos relacionados con la red. Esto puede ser interesante para asignaturas que utilizan la red, como puede ser programación distribuida.
- Mejorar la flexibilidad y compatibilidad de la herramienta permitiendo, por ejemplo, adjuntar también ficheros “.rar”.
- Mejorar la interfaz gráfica en algunos aspectos, como que la elección de herramientas esté clasificada por lenguajes o que solo pidan parámetros extra las herramientas que los necesiten.
- Mantenimiento y mejora de las herramientas existentes, como, por ejemplo, mejorar la herramienta Astyle para que pueda corregir varios ficheros en una misma entrega y haga la media con la calificación.
- Y, por supuesto, la implementación de nuevas funcionalidades relacionadas con procesos de evaluación automática.

6. Bibliografía

[1] K. Mershad, A. Damaj and A. Hamieh, "Using Internet of Things for Automatic Student Assessment during Laboratory Experiments," *2019 IEEE International Smart Cities Conference (ISC2)*, Casablanca, Morocco, 2019, pages 317-323.

[2] Rafael M. Luque-Baena Francisco Luna Luis Arévalo, Francisco J. Rodríguez. Experiencia con una herramienta de pruebas de caja negra para el aprendizaje de asignaturas de programación en evaluación continua. JENUI, pages 61–68, 2017.

[3] Rubén Sebrango Briz. Marco para la evaluación automática de código basado en moodle. SAGEOpen, pages 1–64, 2017.

[4] Guillermo Quintana. Mantenimiento y modularización de EvalCode para su puesta en marcha en plataformas Moodle. 2019

[5] Patil, A. Automatic Grading of Programming Assignments. 2010.

[6] Caiza, Julio C. y Álamo Ramiro, José María del. Programming assignments automatic grading: review of tools and implementations. INTED2013, pages. 5691-5700, 2013.

[7] Moodle Org. Tipo de pregunta CodeRunner. https://docs.moodle.org/all/es/Tipo_de_pregunta_CodeRunner.

[8] Oracle. Virtualbox. https://www.virtualbox.org/.

[9] SSH Communications Security, Inc. SSH Protocol. https://www.ssh.com/ssh/protocol/.

[10] SSH Communications Security, Inc. SFTP – SSH Secure File Transfer Protocol https://www.ssh.com/ssh/sftp/.

[11] The Apache Software Foundation. Apache HTTP Server Project. https://httpd.apache.org/.

[12] Microsoft. Visual studio code. https://code.visualstudio.com/docs.

[13] The Geany contributors. About Geany. https://www.geany.org/about/geany/.

[14] Inc. GitHub. Github. https://github.com/.

[15] Kai Willadsen. Meld. https://meldmerge.org/.

[16] JUnit Team. JUnit. https://junit.org/junit4/.

[17] Oliver Burn. Checkstyle. https://checkstyle.sourceforge.io/.

[18] SourceForge. Artistic Style 3.1. http://astyle.sourceforge.net/.

[19] CS50 Docs. style50. https://cs50.readthedocs.io/style50/.

[20] Firejail. About. https://firejail.wordpress.com/.

[21] CS50 Docs. compare50. https://cs50.readthedocs.io/projects/compare50/en/latest/.

[22] Moodle Org. Acerca de Moodle. https://docs.moodle.org/all/es/Acerca_de_Moodle.

[23] Moodle Org. Directorio Moodle del sitio Moodle. https://docs.moodle.org/all/es/Directorio_Moodle_del_sitio_Moodle.

[24] Moodle Org. Renderer. https://docs.moodle.org/dev/Renderer.

[25] Moodle Org. Módulos de actividades (desarrollador). https://docs.moodle.org/all/es/M%C3%B3dulos_de_actividades_(desarrollador).

[26] Raúl Ivorra Oltra, Sergio Luján Mora. Ampliación de Moodle: Creación de módulo actividad. CUIEET, 2009

[27] Moodle Org. Form API. https://docs.moodle.org/dev/Form_API.

[28] Mario Piattini et al. Mantenimiento del software. Ra-Ma, 2000.

[29] Moodle Org. Developer Mode. https://docs.moodle.org/dev/Developer_Mode.

[30] SELinux Project. About SELinux Wiki. https://selinuxproject.org/page/Main_Page.

7. Anexo. Configuración de red del servidor de pruebas

La correcta configuración de la máquina virtual Linux es de los puntos más importantes a la hora del desarrollo de EvalCode, sobre todo la configuración de red, que es la que permite acceder a Moodle desde fuera de la máquina virtual. Para el desarrollo de este proyecto, se utilizó un servidor de pruebas que nos permitiera desarrollar fuera del servidor de explotación. Este servidor nos permitía desarrollar y probar las nuevas funcionalidades para después integrar estos cambios en el servidor de explotación. Este servidor de pruebas se ejecuta sobre una máquina virtual idéntica a la del servidor de explotación, pero que utiliza distintas configuraciones. El servidor de explotación necesita en su máquina virtual una configuración específica para la red de la Universidad de Cantabria que no es válida para ejecutarlo desde otro ordenador. En este apartado vamos a detallar los pasos que se deben llevar a cabo para la configuración de red específica de este servidor de pruebas.

Estas directrices están enfocadas al uso de una máquina virtual de Ubuntu, la utilizada en el servidor de explotación, corriendo sobre el software VirtualBox. Para esta máquina virtual es importante destacar que hay que reservar un cierto espacio en el disco, ya que puede ocupar entre 20 y 30 GB una vez importada en VirtualBox, mientras que la imagen de la máquina exportada pueden ser unos 12 GB.

7.1. Configuración de un adaptador puente

Un primer paso de la configuración sería conectar la máquina implementando un adaptador puente (*bridge adapter*). Este adaptador permitirá entrar a Moodle desde el navegador y mediante protocolos SSH y SFTP.

Para llevar a cabo la configuración es imprescindible conocer la IP del ordenador anfitrión o *host* de la máquina virtual. Los datos de red se pueden consultar fácilmente en cualquier sistema Linux mediante el comando "*ifconfig*". En la figura 40 se puede ver el resultado de aplicar este comando en el sistema Linux que se utilizará de ejemplo.

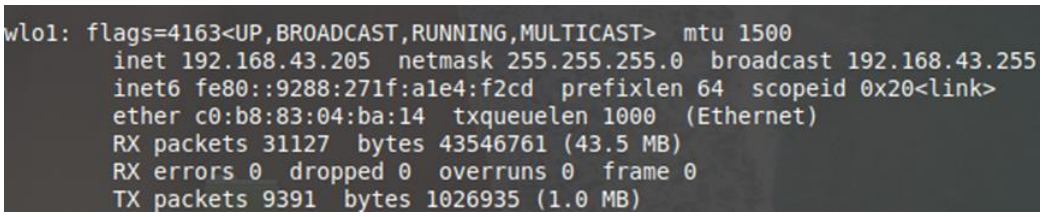


Figura 40: Ejemplo de salida de comando "*ifconfig*"

Conociendo ya los datos de red del sistema operativo *host*, se deben llevar a cabo los siguientes pasos:

En VirtualBox, entrando en opciones de la máquina virtual o *VM settings*, se ha de configurar un adaptador puente como se puede ver en la figura 39, asegurándose de que como nombre se pone la interfaz del *host* que tenga acceso a la red (en este caso, la interfaz Wi-Fi).

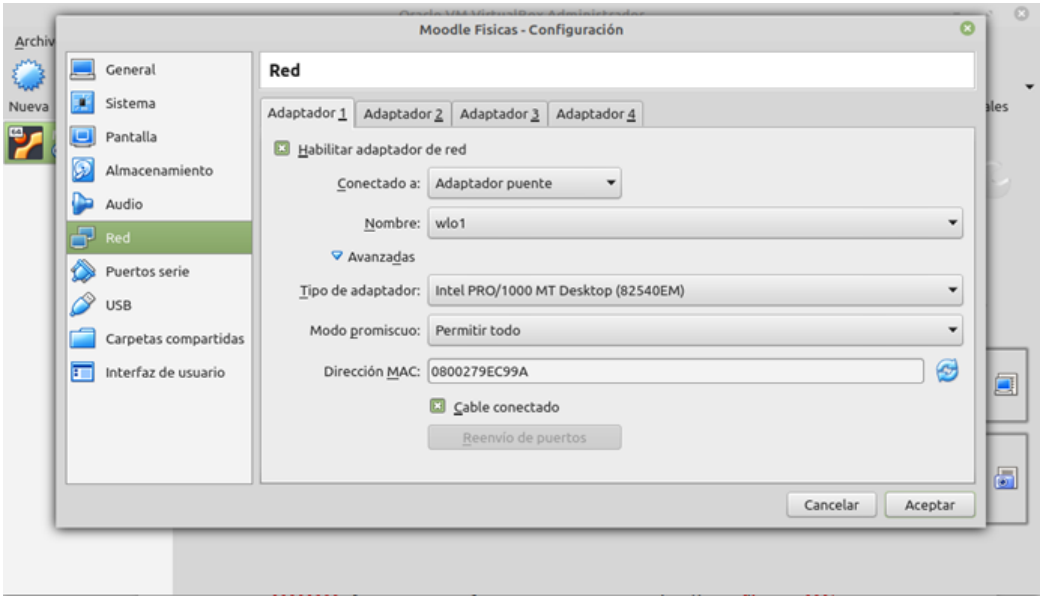


Figura 39: Configuración de un adaptador puente en VirtualBox

El siguiente paso es, en la máquina virtual, cambiar el fichero *"/etc/netplan/50-cloud-init.yaml*" para definir una interfaz de red con IP fija dentro de la red (en este caso, 192.168.43.205). En la figura 41 hay un ejemplo de una configuración de este fichero.

```
network:
   ethernets:
      enp0s3:
         dhcp4: no
         dhcp6: no
         //Siempre igual la IP del so host menos el ultimo dígito
         addresses: [192.168.43.46/24]
         //Como la de arriba pero el ultimo digito siempre en 1
        #gateway4: 192.168.43.1
         nameservers:
            addresses: [212.166.210.82, 212.166.132.104] //Estas pueden variar
         optional: true
   version: 2
```

Figura 41: Ejemplo de definición de una interfaz de red con IP fija

Además, para acceder a Moodle desde el navegador, en el sistema operativo *host* Linux se debe modificar el fichero "*/etc/hosts*" y añadir la línea "192.168.43.46 moodle.ctr.unican.es", como se ve en la figura 42.

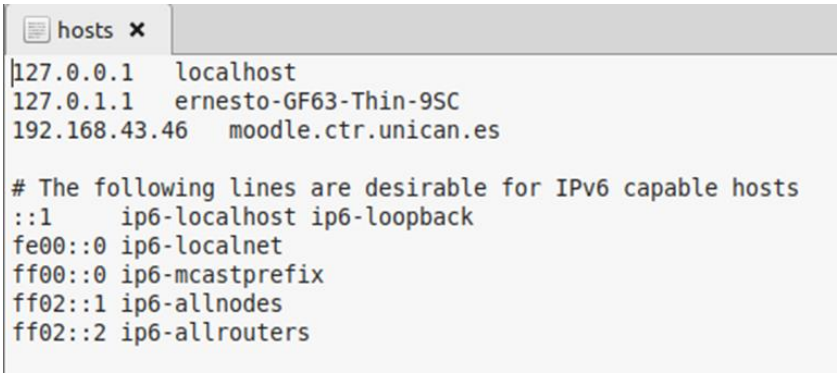


Figura 42: Ejemplo de fichero "*/etc/hosts*"

Para aplicar los cambios se puede ejecutar el comando "*netplan apply*" o reiniciar la máquina virtual.

Con esta configuración se puede acceder al Moodle de las siguientes maneras:

- **Acceso web a Moodle**:
  Abrir un navegador e ir a la dirección web "https://moodle.ctr.unican.es/".

- **Acceso gráfico al sistema de ficheros**:
  Abrir un gestor de ficheros e ir a "sftp://moodle@192.168.43.46/".

- **Conectar por SSH desde la *shell* del sistema operativo *host* a la VM**:
  En la *shell* del sistema operativo *host* escribir "ssh moodle@192.168.43.46".

  De esta manera se pueden hacer algunas operaciones de manera más cómoda que directamente en la máquina virtual ya que esta, por ejemplo, no permite pegar un texto que se ha copiado en el sistema operativo anfitrión, cosa que si se puede hacer conectando por SSH desde el *host*.

7.2. Configuración de un adaptador NAT

La configuración del adaptador puente permite conectarse a Moodle desde el navegador y acceder mediante los protocolos SSH y SFTP a la máquina, pero no proporciona conexión a internet dentro de la máquina virtual, la cual es necesaria para instalar programas.

Para tener red dentro de la máquina hay que habilitar un segundo adaptador junto al adaptador puente, esta vez un adaptador de tipo NAT (*Network Address Translation*). En

la figura 43 se puede ver un ejemplo de cómo se añade este adaptador a los ajustes de VirtualBox.

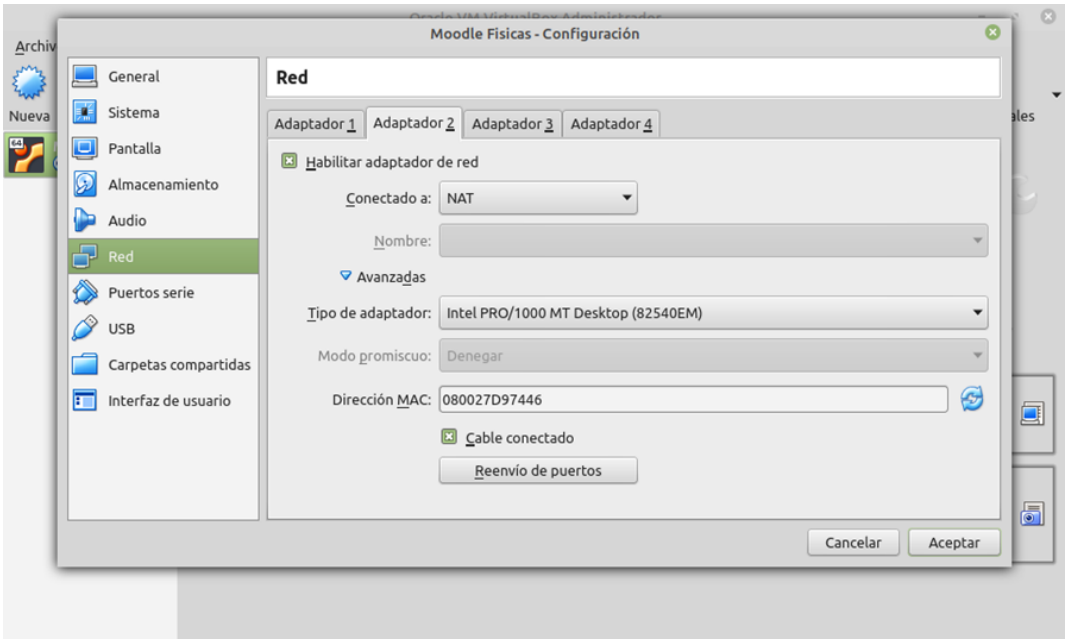


Figura 43: Configuración de un adaptador NAT en VirtualBox

A continuación, en el fichero "*/etc/netplan/50-cloud-init.yaml*" se debe definir otra interfaz de red, en este caso "enp0s8", pero ahora con IP dinámica.

```
 GNU nano 2.9.3                    /etc/netplan/50-cloud-init.yaml

# This file is generated from information provided by
# the datasource.  Changes to it will not persist across an instance.
# To disable cloud-init's network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
    renderer: networkd
    ethernets:
        enp0s3:
            addresses: [192.168.43.46/24]
            dhcp4: no
            dhcp6: no
#            gateway4: 192.168.43.1
            nameservers:
                addresses: [8.8.8.8, 8.8.4.4]
            optional: true
        enp0s8:
            dhcp4: true
            dhcp6: true
    version: 2
```

Figura 44: Estado final del fichero *"/etc/netplan/50-cloud-init.yaml*"

Tras habilitar este adaptador NAT ya se debe tener internet dentro de la máquina. En la figura 44 se puede ver el estado final del fichero "*/etc/netplan/50-cloud-init.yaml*" con

las dos interfaces de red configuradas. Adaptando la IP fija que se utiliza para el adaptador puente a la IP del sistema operativo *host* como se ha indicado en la figura 41, este proceso debería ser válido para cualquier sistema operativo Linux.